Symbolic Inference Methods for Databases

Slawomir Staworko

Université de Lille 1
Centre de Recherche en Informatique, Signal et Automatique de Lille

Habilitation à Diriger des Recherches

spécialité « Informatique »

Méthodes d'Inférence Symbolique pour les Bases de Données

Symbolic Inference Methods for Databases

par

Sławek Staworko

Habilitation soutenue publiquement le 14 décembre 2015 devant le jury composé de :

M.	James Cheney	Université d'Édimbourg (UK)	(Rapporteur)
M.	Laurent Grisoni	Université de Lille 1	(Examinatuer)
M.	Amaury Habrard	Université Jean Monnet Saint-Etienne	(Rapporteur)
M.	Jean-François Raskin	Université Libre de Bruxelles	(Examinatuer)
M.	Jean-Marc Talbot	Université d'Aix-Marseille	(Rapporteur)
Mme.	Sophie Tison	Université de Lille 1	(Directrice)

N° ordre : 41969

Résumé

Ce mémoire est une courte présentation d'une direction de recherche, à laquelle j'ai activement participé, sur l'apprentissage pour les bases de données à partir d'exemples. Cette recherche s'est concentrée sur les modèles et les langages, aussi bien traditionnels qu'émergents, pour l'interrogation, la transformation et la description du schéma d'une base de données. Concernant les schémas, nos contributions consistent en plusieurs langages de schémas pour les nouveau modèles de bases de données que sont XML non-ordonné et RDF. Nous avons ainsi étudié l'apprentissage à partir d'exemples des schémas pour XML non-ordonné, des schémas pour RDF, des requêtes *twig* pour XML, les requêtes de jointure pour bases de données relationnelles et les transformations XML définies par un nouveau modèle de transducteurs arbre-à-mot.

Pour explorer si les langages proposés peuvent être appris, nous avons été obligés d'examiner de près un certain nombre de leurs propriétés fondamentales, souvent souvent intéressantes par elles-mêmes, y compris les formes normales, la minimisation, l'inclusion et l'équivalence, la cohérence d'un ensemble d'exemples et la caractérisation finie. Une bonne compréhension de ces propriétés nous a permis de concevoir des algorithmes d'apprentissage qui explorent un espace de recherche potentiellement très vaste grâce à un ensemble d'opérations de généralisation adapté à la recherche d'une solution appropriée.

L'apprentissage (ou l'inférence) est un problème à deux paramètres : la classe précise de langage que nous souhaitons inférer et le type d'informations que l'utilisateur peut fournir. Nous nous sommes placés dans le cas où l'utilisateur fournit des exemples positifs, c'est-à-dire des éléments qui appartiennent au langage cible, ainsi que des exemples négatifs, c'est-à-dire qui n'en font pas partie. En général l'utilisation à la fois d'exemples positifs et négatifs permet d'apprendre des classes de langages plus riches que l'utilisation uniquement d'exemples positifs. Toutefois, l'utilisation des exemples négatifs est souvent difficile parce que les exemples positifs et négatifs peuvent rendre la forme de l'espace de recherche très complexe, et par conséquent, son exploration infaisable.

Abstract

This dissertation is a summary of a line of research, that I was actively involved in, on learning in databases from examples. This research focused on traditional as well as novel database models and languages for querying, transforming, and describing the schema of a database. In case of schemas our contributions involve proposing an original languages for the emerging data models of Unordered XML and RDF. We have studied learning from examples of schemas for Unordered XML, schemas for RDF, twig queries for XML, join queries for relational databases, and XML transformations defined with a novel model of tree-to-word transducers.

Investigating learnability of the proposed languages required us to examine closely a number of their fundamental properties, often of independent interest, including normal forms, minimization, containment and equivalence, consistency of a set of examples, and finite characterizability. Good understanding of these properties allowed us to devise learning algorithms that explore a possibly large search space with the help of a diligently designed set of generalization operations in search of an appropriate solution.

Learning (or inference) is a problem that has two parameters: the precise class of languages we wish to infer and the type of input that the user can provide. We focused on the setting where the user input consists of positive examples i.e., elements that belong to the goal language, and negative examples i.e., elements that do not belong to the goal language. In general using both negative and positive examples allows to learn richer classes of goal languages than using positive examples alone. However, using negative examples is often difficult because together with positive examples they may cause the search space to take a very complex shape and its exploration may turn out to be computationally challenging.

Acknowledgements

The freedom to pursue any research question is a luxury that comes with a number of responsibilities. One of them is honesty: being honest to oneself and to the community at large. This in turn calls for humility, seeing things the way they are and not the way we wish them to be. But it wouldn't be a luxury if there wasn't some joy involved. Honesty, humility, and joy are concepts that we understand almost instinctively and yet we need their examples to inspire us. I was lucky to have found them in many of my close colleagues, in particular Sophie Tison, Peter Buneman, Rémi Gilleron, Joachim Niechren, and Marc Tommasi.

Research can be a fascinating journey that is best made in the company of others but at times it takes us to remote areas of solitary questioning of matters that few understand and even fewer appreciate. It can be an alienating activity that caries the risk of crossing the point of no return, and inadvertently, becoming a hermit. Thankfully, I have been gifted with a group of close friends who have supported and anchored me through many stormy times and constantly remind me that not every endeavor in life needs to be exemplary. I'm particularly grateful for Maciek Falski, Samuel Hym, Sebastian Maneth, and Tom Sebastian.

This dissertation has been written during my fellowship at University of Edinburgh while on a leave of absence from University of Lille 3, with the support of the laboratory CRIStAL (UMR 9189) and the DIACHRON FP7 project (EU grant no. 601043). A number of sections have been drafted in the beautiful Scottish Highlands, which I could enjoy in October 2015 thanks to the warm hospitality of Peter Buneman.

– Która godzina? – Tak, jestem szczęśliwa . . .

Na wieży Babel, Wisława Szymborska, 1962.

Dans la vie, rien n'est à craindre, tout est à comprendre.

Maria Skłodowska-Curie, 1867–1934.

He's a Frankenstein! And they're all alike. It's in their blood. They can't 'elp it. All those scientists, they're all alike. They say they're working for us. What they really want is to rule the world!

Young Frankenstein, Mel Brooks, 1974.

Contents

1 Introduction

Storing and processing information is crucial in virtually any human enterprise, and throughout history a number of technologies have been developed with the purpose of assisting us in this activity. While many information technologies ultimately benefit humanity, they are inherently artificial and unnatural, and pose challenges and obstacles to their use.

Take for instance writing, strictly speaking a technology of representing a language with an established set of symbols, introduced to overcome the limits to the amount of data that human memory can reliably retain [Gau85]. The emergence of writing is so fundamental that it defines the beginning of history as opposed to prehistory. Writing gave birth to logic not only because it introduced symbols but also because it has greatly influenced the way we think and reason [Ong86]. We might have difficulty recognizing it because it is a skill that we deeply internalize and tend to take for granted. However, it was not until 19th century when through compulsory education literacy became widespread whereas before it used to be a skill reserved to elites, often a profession on its own, *inaccessible* to general public because requiring learning. Despite the early concerns that writing is a technology that weakens the mind, by reliving it of work that keeps it strong [Ong02], writing turned out to be consciousness-raising and ultimately humanizing. Still, one of the early criticisms of writing remains valid: it is *unresponsive*. Unlike a dialogue with a real person, it is impossible to get an answer to a question from a written text unless the answer is explicitly spelled out in the text.

Much of what has been said about writing can and has been said about computers. Computers are a technology that allows to process data in a manner more efficient and reliable than humans, and their emergence has marked the Digital Age. Apart from improving our efficiency they also enable our creativity [Edw01] despite early concerns that they too would weaken the mind. Because computers are machines that follow precisely demarcated lines of execution, they also affect the way we structure our thoughts [Pea85], and naturally, using them requires learning and expertise, which to this day remains an important *accessibility barrier*. For the same reasons, computers require precisely structured input, and while not precisely unresponsive, a well known and ever present phenomena of *Garbage In, Garbage Out* [Bab64, LHB10] further demonstrates the difficulty that many users face when using computers.

The source of the difficulty in using a technology is a complex problem, and often, as it is the case with information technologies, it is know to involve human factors such as age, education, and the level of openness to change [GCS12]. We, however, focus on the factors coming from the side of technology. A technology is often a solution to a concrete problem conceived with a narrow target group of highly-qualified users, in many cases the architects of the solution themselves. As the technology proves to have a wider range of applications, the group of users broadens and often includes those without the necessary technical background, which creates an accessibility barrier. In the specific context of computer-based information systems, we focus on the aspects of the barrier originating from the manner of representing and processing information by machines, which is very different from how human memory and reasoning work [WR09]. Consequently, a human user wishing to satisfy an information need is required to formulate it in a language that is artificial and foreign. What illustrates the unnatural character of this process is that it is typically a one way street: the user alone is responsible for making the information need understood and a failure results in incorrect results (Garbage In, Garbage Out). This stands in great contrast to how humans communicate and exchange information [Tay68]: the effort to get the message across is shared between the participating parties and the communication follows natural protocols to ensure mutual understanding. Hence, a reasonable way of lowering the accessibility barrier is to enable computers understand the information needs of a user from input other that a formal language and possibly in a way involving interaction.

The ability to understand possibly complex concepts is generally considered to be a hallmark of intelligence and while intelligence does not have a precise definition, the Turing test [Tur50] is one of the earliest attempts to define intelligent computer behavior. Basically, it is a challenge of making computers exhibit intelligence by the ability to carry a conversation in a manner indistinguishable from the way humans converse. As thought provoking as it is, it has also been widely criticized on the grounds that mimicking (certain aspects of) intelligent behavior is not necessarily a proof of intelligence [RN10]. A close look at the recent success at passing the Turing test [WS15] further demonstrates the ineptness of the test to confirm intelligence: a chatterbot pretending to be a foreign child is likely to inspire forgiveness for its grammatical errors and lack of general knowledge, and as a result, may pass the test. In order to isolate concerns, we remove linguistic aspects from our considerations and limit our research to much simpler user input in the form of examples. We point out, however, that querying databases using natural language interfaces is an active research topic [ART95] with some promising advances made recently [LJ14].

Traditionally, an information system consisted of a *relational database* [RG00], or less commonly an object-oriented database, that was responsible for storing and processing the data in accordance with rules formalized by its *architects* who would also study the information needs of a target group of *users* and devise appropriate *interfaces* to address those needs. However, with the ever growing amount of data available to practically everyone, as evidenced for instance by the Open Data initiative, it is virtually impossible to conceive all possible uses of data and for pragmatic reasons users are given direct access to either the complete database or its partial view. Because this data often needs to be loaded into other databases or applications, for interoperability reasons XML is used. While being a human-readable data representation language, XML has quickly become the de facto standard for data exchange between applications[AL05] and a prime example of *semi-structured* data model [Flo05]. Furthermore, the movement of democratization of data, where everyone can not only produce data but also describe its relationship to data available elsewhere, has led to an increased interest in graph database models and the emergence of appropriate data formats such as RDF [AGP09, APR+11]. Interestingly, as the data models evolve to allow easier access to a broader range of users, the architects become less involved in designing the databases to accommodate potential information needs. This tendency can be illustrated with the use of schemas whose conception is one of the tasks of a database architect. While relational databases come with precise schema that provide a description of the database structure and the intended semantics of the data, in semi-structured data models such as XML the schema is optional, and from its very conception RDF has been proposed as a schema-free data format to promote its wide-spread use. Our research spans over the above three major database models, each presenting particular challenges and opportunities for assisting users in satisfying their information needs.

The information needs of a user vary with application domain and the type of data that the user has access to. In our work we have investigated 3 types of formalisms that can be used to address some of the information needs. *Querying* is one of the most common tasks performed on databases and in its basic form it consists of identifying data objects relevant to the user's search. *Reformatting* is one of the simplest data *transformation* tasks but it already introduces a significant challenge when attempting to assist a user. Finally, in many novel scenarios the databases are schema-free and the user might wish to know the *schema*, which can be seen as a *summary* of the structure of the database. Here, we first needed to propose and study an appropriate schema formalism before investigating how to construct it for a given database.

To approach the problem of making computers understand the information needs of a user, we follow the steps of the seminal framework of *language identification in the limit* [Gol67], known also as the framework *learning from examples* and belonging to the group of *grammatical inference* methods. The framework is based on the interaction between two actors, the *learner*

and the *instructor*, where the instructor attempts to convey a *concept* (essentially, a language) following a precisely defined protocol. A number of possible protocols has been considered and we focus on a the most fundamental one that permits the instructor to present *examples* to the learner. The instructor might be restricted to use only *positive* examples, which illustrate what the goal concept is, or may also be allowed to use *negative* examples, that illustrate what the goal concept is not. Ideally, the process continues until the learner infers the correct concept, and a class of concepts is *learnable* if for every concept from the class, the process terminates after a finite number of steps. While initially no bound was imposed on the number of examples or the computational resources that the learner use on inferring the goal concept, the framework has been eventually amended to account for tractability [OG91, dlH97]. Essentially, the learner is required to work in time polynomial in the total size of the examples presented so far and the instructor is required to be able to convey every concept with a number of examples whose total size is bounded by a polynomial.

A natural question is whether the learning framework outlined above is appropriate for studying the problem of a computer (the learner) being able to proactively understand information needs of a user (the instructor). Indeed, the interaction between the computer and the user is basically restricted to *yes-or-no* questions, which may be too limited to fully understand the complexity of the problem at hand, and perhaps a framework based on a richer, more expressive protocols would be more suitable. Take for instance the framework of *learning with queries* [Ang87, Ang88, AAP+13], where the learner presents a hypothesis, basically a concept that the learner believes to be the goal concept, and the instructor may accept it, at which point the process terminates, or reject it by presenting a *counterexample*, an example that differentiates the proposed hypothesis from the goal concept, and the process continues. This framework allows a richer form of interaction and we take inspiration in it when investigating aspects of interaction in learning. However, this framework also requires the user to understand the formal language defining concepts, which is opposed to the very motivation of our research: the user may be unfamiliar or unable to use this formal language. On the other hand, the framework of learning from examples is applicable to very natural interaction scenarios, for instance where the user interacts with the system using only a pointing device. We also point out that the simplicity of the framework of learning from examples makes it fundamental, and in fact a large number of results on learnability established within this framework has immediate implications in any framework that subsumes it, including the framework of learning with queries. Finally, we remark that a variant of the Turing test known as the *minimum intelligent signal test* [McK97] is similarly restricted to yes-or-no questions, which further validates our approach. To conclude, while the simplicity of the interaction in the proposed framework may address the original problem only partially, it does, however, capture conceivable practical scenarios, and furthermore allows to identify a number of fundamental opportunities and limitations inherent to the problem at hand.

The main research question that we pursue is the learnability of the various languages for expressing information needs of a user. The question has two parameters: 1) the precise class of languages and its expressive power and 2) the richness of the input that the user can provide: positive examples alone are less informative than positive and negative examples. Our results can be distilled in a this general principle: *The more expressive the class of languages is and the less informative the user input is, the less feasible learnability is.* Naturally, there are exceptions to this rule because learning formalism of restricted power from very informative input often boils down to solving complex set of constraints which proves learning unfeasible. Furthermore, we point out that sometimes using negative examples can be difficult because intuitively they do not provide sufficiently detailed information on why a given example should not be recognized by the goal language.

Investigating learnability of a class of languages requires a thorough understanding of the class

of languages and pushes us to study a number of fundamental problems, often of independent interest. Languages are typically represented with expressions/formulas that follow a formal grammar, which might allow multiple expressions to define the same language. Because an algorithm needs to operate on expressions, it is beneficial to identify *equivalent* expressions and represent the corresponding language with a *canonical* representative, which calls for defining normal forms and studying *normalization* and *minimization* of expressions.

Learning is a function problem that takes a set of examples and returns an expression representing a language from a given class. However, there might not exists a language *consistent* with the given set of examples and the learning algorithm should indicate such a situation, which might be due to erroneous input or weak expressive power of the class of languages. Testing *consistency* of the input sample i.e., checking if there exists an expression consistent with the given examples, is a decision problem that allows us to establish lower bounds of learnability and helps us to identify a number of settings for which learnability is not feasible.

Learning can be viewed as an exploration of a potentially large *search space* of languages with the examples being clues or landmarks that lead us towards the appropriate solution. This search space is organized by the standard language *containment* relation, and understanding this relation on the level of expressions is essential for properly navigating this search space. In our work we aim to identify classes of languages for which the containment of languages can be characterized with structural means on the corresponding expressions. This typically leads to defining *generalization operations* that operate on expressions and organize the search space in a manner very close, if not identical, to how this space is organized by the containment relation. Our learning algorithms are driven by generalization operations and the way they seek the appropriate solution can be divided into two categories depending on weather or not negative examples are used.

In the presence of positive examples only, our learning algorithms generally attempt to identify the *minimal fitting expression* i.e., the most specific expression that is consistent with the positive examples. One way to identify such an expression is to begin with a *universal* formula i.e., a formula that accepts all examples, and iteratively specialize it under control of positive examples i.e., making sure that the expression always accepts all positive examples, thus avoiding overspecialization. We point out that specialization is the inverse of generalization and is done with inverse application of generalization operations. We point out that depending on the class of formalisms the most specific formula might be large or need not be unique. However, even in such a situation the specialization approach might be appropriate for learning.

When negative examples are available, learning is done in two phases. First, we construct a minimal expression fitting the positive examples while not satisfying any of the negative examples. In the next phase, we iteratively generalize the previously constructed formula while ensuring it does not satisfy any of the negative examples until no further generalization can be performed. In a manner of speaking the negative examples are used to *control* the generalization process and learning is feasible if the required number of negative examples can be maintained within the (polynomial) limits imposed by the framework. The issue of identifying the necessary negative examples to control generalization leads us to a closely related and very interesting research question of *characterizability* [DP92, dlH97, Dal00, tCKT10]: given a formula is there a set of positive and negative examples that characterizes the given formula within the given class of formalisms i.e., the given formula is the only formula (up to equivalence) in the class of formalisms that is consistent with the set of examples. And if it is the case, how large the set of examples needs to be.

We then investigate aspects of *interaction* in the context of learning queries inspired by the framework of *Minimally Adequate Teacher* [Ang87]: rather than to have the user select and label data objects as positive or negative examples we wish the computer to select the data objects for

the user to label in a manner that ensures fast convergence towards the goal concept. This path of research leads us to quantify and measure the information about the goal query that a given example can provide (in the context of previously labeled examples). Also, this work leads to developing a more general framework for learning queries for *Big Data*.

Finally, learning *transformations* adds a particular flavor to our considerations. Since transformation is a function, an example is a pair consisting of the input and the output. As such it contributes both positive information – the given input is transformed into the given output – and negative information – the given input is not transformed into any other output. Consequently, we do not consider the examples to be either positive or negative but simply examples. The transformations we study are defined using *tree-to-word transducers*, finite state machines with output. Learning transducers requires us to study a number of fundamental issues: normalization, minimization, testing equivalence, all which are elegantly expressed with a *Myhill-Nerode* theorem. The proof of the Myhill-Nerode theorem is essentially an effective procedure for constructing a *canonical* representative of a given transducer. This procedure is adapted to produce the canonical transducer from input that characterizes the goal transducer with a set of examples. However, this learning algorithm returns the goal transducer only if the input set of examples is sufficiently informative, otherwise the algorithm may choose to *abstain* if it fails to construct a transducer consistent with the input sample although such a transducer may exists. Allowing the algorithm to *abstain* from providing an answer seems a reasonable compromise because testing consistency of an arbitrary input set of examples turns out to be intractable.

A detailed list of relevant contributions follows:

1. We considered the novel model of *Unordered XML* [ABV12], proposed a suitable language of schemas and studied its basic properties [1], and investigated the learnability of its subclasses [5].
2. We proposed *Shape Expression Schemas* (ShEx) for RDF [11, 7] developed in collaboration with W3C [W3C13b] and we report preliminary ideas on learning ShEx.
3. We investigated learnability of *twig queries* for XML [13]. Furthermore, we studied the problem of characterizability of twig queries [14] that required a study of injective embeddings of twig queries [10].
4. We studied *interactive learning* of join queries for relational databases [3, 4] and proposed a paradigm for interactive query learning on *Big Data* [2].
5. We proposed a model of *sequential tree-to-word transducers* (STW) for modeling XML transformations suitable for reformatting [8, 12] and studied its learnability [9]. This required proposing a normal form and an effective methods *normalization* and *minimization*. It is a significant result because the existence of normal forms for general tree transducers is a long-standing open question [Eng80].

Related work. In the recent years we have witnessed a proliferation of research on learning methods in databases [AAP$^+$13, BGNV10, BCL15, LMN10, tCDK13]. The framework of learning in the limit [Gol78] has inspired many approaches including learning Document Type Definitions (DTDs) and XML Schemas [BNV07, BNSV10, BGNV10], two most wide-spread schema languages for XML [GM11, BNVdB04], learning XML transformations [LMN10], learning n-ary automata queries for XML [CGLN07], learning (k, l)-contextual tree languages [RBVdB08] used as wrappers for web information extraction, and learning regular path queries for graphs [BCL15]. Cohen and Weiss propose an alternative approach to assist a user in querying XML and graph databases [CW13, CW15]: rather than constructing a single query they take inspiration in the possible world semantics of incomplete databases [Lip79] and compute *possible* and *certain* answers defined over the set of all queries that are consistent with the given set of positive and

negative examples. We point out, however, that also this approach suffers from high complexity due to the intractability of the consistency problem. XLearner [MKM04] is a practical system for learning XQuery [W3C07b] with equivalence and membership queries and based on DFA inference algorithm proposed by Angluin in [Ang87]. It is worth to point out that twig queries are known not to be learnable with equivalence queries alone [CCG06]. The problem of learning query has been studied in the setting of relational setting [SPGMW10, TCP09, GVdB09, AAP$^+$13, ZEPS13] but the expressiveness of relational query languages renders learning quickly unfeasible. Learning quantified Boolean formulas for the nested relational model has been studied [AAP$^+$13, AHS12] in the framework of learning with membership queries [Ang88]. Learning of relational queries is closely related to the problem of *definability* [Ban78, Par78] that uses automorphisms of relational structures to identify pairs of relational instance such that one is a view of another. An alternative model of *Probably Approximately Correct* (PAC) learning [Val84] has been employed for learning relational schema mappings [GS10, QCJ12, tCDK13, AtCKT11a, AtCKT11b].

The problem of characterizability and interactive learning are closely related to the problem of *teaching* [GRS93, GK95, SDHK91, SM91, ABCST92] where the set of examples to be presented is selected by a helpful teacher, and the objective is to identify the least amount of necessary examples. Characterizability is also closely related to *learning and verification* of queries [AAP$^+$13], where examples are generated in order to verify that a given query is the one intended by the user. A number of notions of characterizability has been studied in the context of grammatical inference [dlH97]. Communication complexity [Kus97] studies the lower and upper bounds on the amount of data send over communication channels by agents trying to accomplish a common task such as verifying they have the same data structure. While the interactive learning scenario aims at minimizing the number of interactions, this does not necessarily correspond to minimizing the amount of data being shared.

DTDs under *commutative closure* [BM99, NS] have been considered as schemas for Unordered XML. However, this approach suffers from high complexity: testing membership under commutative closure is NP-complete [KT10]. A way to circumvent this problem is to consider DTDs defining a commutatively-closed language but verifying this property turns out to be PSPACE-complete [NS]. A number of approaches for validation of RDF has been previously proposed. While RDF Schema (RDFS) is typically employed as a light ontology language, the formal specification [W3C12] mentions the possibility of using it as a schema language, alas without formalizing this usage. OSLC Resource Shapes (ResSh) [RLHS13] essentially extend RDFS by allowing to specify cardinality constraints ?, $*$, $+$, and 1 on types which makes its semantics very close to the one of ShEx but we point out that ResSh assumes the typing to be given with `rdf:type` predicates while the intention of ShEx is to construct a valid typing, a much more challenging task. Although the ontology language OWL [W3C04] can enforce certain constraints, they are limited in power because OWL is interpreted under open world assumption (OWA) and without unique name assumption (UNA). Consequently, a modified semantics for OWL has been proposed [Sir10] that uses OWA when inferring new facts while employing closed word assumption (CWA) and UNA when enforcing constraints, however concerns have been raised [RLHS13] about the potential confusion arising from the mixed semantics. While [TSBM10] outlines a method of translating OWL constraints into SPARQL queries, the size of the resulting SPARQL queries seems to depend on the size of the OWL constrains, and it remains an open problem if such an approach renders validation feasible. Similar criticism applies to other solutions based on SPARQL queries, such as SPIN [BGR12], while they are very powerful and expressive, their use may require significant computational resources. There exists a host of work on graph summarization, which is related to inference of schemas for RDF. The goal of summarization is to compute compact but accurate representation of an input graph and considered approaches include collapsing nodes by means of graph simulation [FLWW12, ZDYZ14, KC10, TLR13], estimating the frequency with

which subgraphs match given query patterns [MASS08], or aggressively fusing edges with the same source and/or target[CGM15].

The problems of normalization, minimization, Myhill-Nerode characterization, and inference are fundamental problems that have been studied for a large number of classes of transducers. The normal form of STWs is inspired by the normal forms for functional rational transformations [RS91], deterministic top-down transducers [EMS09], as well as the normal form for various classes of deterministic string-to-string transducers (cf. [Cho03] for a survey). Existence of normal forms for more complex classes such as Macro Tree Transducers is a long-standing open problem [Eng80].

Organization. This document is organized as follows. In Section 2 we present an outline of learning theory. In Section 3 we present a novel schema formalism for Unordered XML together with result on its learnability, and additionally present a novel schema formalism for RDF and outline initial ideas for learning schemas for RDF. In Section 4 we describe learning and characterization of twig queries for XML. In Section we present results on interactive learning of join queries for relational databases. In Section 6 we present sequential tree-to-word transducers, describe their normal form and a normalization algorithm, present a Myhill-Nerode theorem for this class of transducers, and present a learning algorithm for them. Finally, in Section 7 contains final conclusions and the discussion of future work.

2 Fundamentals of learning from examples

In this section we present the fundamentals of *learning from examples* with a light historical overview while illustrating the introduced concepts on a running example of learning *bag expression*, which we reuse later on in schemas for Unordered XML and schemas for RDF (Section 3). We begin by describing the *learning setting* that precisely defines the class of languages, their representation, and type of examples used. Next, we recall the framework of *learning in the limit*, describe a learning method of *minimally fitting expression*, and present one of the fundamental limitation of learning from examples. We then move to the framework of *learning in polynomial data and time* that addresses the weakness of the previous framework in controlling the amount of computational resources used for learning. This framework imposes strict restrictions on the complexity of learning and allows to identify class of languages that are not learnable with the help of the complexity analysis of the *consistency problem*. Next, we describe an important conceptual tool of *search space* and our approach of learning by exploring the search space with *generalization operations*. The search space also allows us to approach a closely related problem of *finite characterizability*. Finally, we consider *interactive learning*, where the learning algorithm presents data objects for user's consideration and attempts to learn the goal language from user responses.

2.1 Learning setting

Before we can study learnability of a class of languages, we need to define precisely not only the class of languages but also how they are represented with expressions, the examples we shall use, and also how to measure the size of expressions and examples.

Bags. We intend to study learnability of simple classes of bag languages. A *bag* is a generalization of a set that allows the same element to be present a number of times but we also view bags as words where the relative order of symbols is removed. A bag language is a set of bags. We assume a fixed but possibly infinite set of symbols Σ and for the purposes of the presented examples we assume that Σ contains at least 3 symbols a, b, and c. We use a dedicated notation for bags, for instance by $\{\!|a, a, b, c, c, c|\!\}$ we denote a bag with 2 occurrences of the symbol a, one occurrence of b and 3 c's. The *bag union* adds occurrences of all symbols: $\{\!|a, c, c|\!\} \uplus \{\!|a, b|\!\} = \{\!|a, a, b, c, c|\!\}$. Because we use bags to represent the neighborhoods of a node in structured data (unordered trees and graphs), every element of a bag corresponds to a neighboring node, and therefore, we use a unary representation of bags and define its size as the sum of occurrences of all its elements. We point out, however, that a binary representation is also possible and does not change any of the presented results if use properly.

Bag expressions. We use simple subclass RBE_0 of regular bag expressions [11] to define bag languages. Essentially, we use *unordered concatenation* $\|$ of atoms of the form x^0, x^1, $x^?$, x^+, and x^*, with $x \in \Sigma$ and the exponent indicating the admissible number of occurrences: 0 none, 1 exactly once, ? at most once, + any positive number of times, and $*$ an arbitrary number of times. For instance, $a^1 \,\|\, b^? \,\|\, c^*$ defines the language of all bags that have precisely one a, at most one b, and an arbitrary number of c. If a symbol is not present in an expression, then implicitly the exponent 0 is assumed i.e., no occurrence of the symbol in the bag is allowed. For instance, $a^* \,\|\, b^*$ defines bags with arbitrary numbers of a's and b's but no c's whatsoever. The same symbol can be used by multiple atoms and we interpret the $\|$ operator with bag union: for instance, $a^1 \,\|\, a^? \,\|\, b^1 \,\|\, b^+$ defines the language of bags that one or two a's and at least two b. An RBE_0 expression is *single-occurrence* (SORBE_0) if every symbol is used in at most one atom. Naturally, SORBE_0 is strictly less expressive than RBE_0. Additionally, we consider a richer class dRBE_0

that allows concatenations of disjunction of atoms e.g., $a^? \parallel (b^1 \mid c^+)$ defines the language of bags having at most one a and either a single occurrence of b or one or more occurrences of c's.

Representation. The languages that we learn are usually infinite but typically have a finite *representation* based on expressions (or more generally grammars). There are subtle differences between identifying a language and identifying a representation of a language that arise when a language may have multiple *equivalent* representations. Because we wish to learn languages while representing them with expressions, to circumvent the potential problems we make the deliberate effort to *normalize* the space of expressions used to represent the languages.

The expression in SORBE_0 are essentially normalized because $\parallel$ is commutative and we can represent SORBE_0 expressions with sets of atoms: $a^1 \parallel b^? \parallel c^*$ is represented by $\{a^1, b^?, c^*\}$. The class RBE_0 has a natural normal form where each symbol is used exactly once but (nonempty and possibly unbounded) intervals specifying the allowed numbers of occurrences are used, for instance $a^1 \parallel a^? \parallel b^1 \parallel b^+$ can be represented as $a^{[1;2]} \parallel b^{[2;+\infty]}$. Naturally, choosing a representation also comes with a *size* function that plays an important role when tractability aspects of learning are concerned. For the size of an RBE_0 expression is the minimal number of basic (without intervals) atoms necessary to express it e.g., $a^{[1;2]} \parallel b^{[2;+\infty]}$ is equivalent to $a^1 \parallel a^? \parallel b^1 \parallel b^+$ and therefore its size is 4.

Consequently, whenever we study learnability of a class of languages, we introduce it with a (normalized) class of expressions, and treat these two as interchangeable. We point out, however, that the choice of representation may have dramatic effect on learnability, for instance, while regular languages represented with deterministic finite automata are learnable from examples [Gol67, OG91], they are not when represented with nondeterministic finite automata [Gol78]. Part of the reason for this is that deterministic finite automata have normal forms (unique minimal automata) which their nondeterministic analogues do not have [JR93].

Examples. The *examples* we use for learning bag languages are bags and they can be used as *positive* examples i.e., bags that are to be present in the goal bag language, or *negative* examples i.e., bags that are to be absent in the goal language. If a setting using both positive and negative examples is considered, we use $+$ and $-$ to differentiate between examples but if only positive examples are used, we do not use any additional markings. For instance, positive and negative examples of $a^? \parallel b^* \parallel c^1$ include

$$(+, \{\!|a, c|\!\}),\ (-, \{\!|a, b|\!\}),\ (+, \{\!|b, b, c|\!\}),\ (-, \{\!|a, a, c|\!\}),$$

and all positive examples of $a^1 \parallel b^? \parallel c^?$ are

$$\{\!|a|\!\},\ \{\!|a, b|\!\},\ \{\!|a, c|\!\},\ \{\!|a, b, c|\!\}.$$

A set of examples is *consistent* with a language (or an expression) if the language contains all positive examples and none of the negative ones.

2.2 Learning in the limit

In [Gol67], Gold proposed an online, incremental learning framework where an algorithm is presented with examples and after each example it output a hypothesis. We illustrate this framework on an example of learning $a^? \parallel b^*$ from positive and negative examples by a possible learning algorithm whose inner workings are left unspecified. The first presented example is $(+, \{\!|a, b, b|\!\})$ and the algorithm proposes $a^* \parallel b^*$. The second example is $(-, \{\!|a, a|\!\})$ and the algorithm refines the hypothesis to $a^1 \parallel b^*$. The third example is $(+, \{\!|b|\!\})$, and the subsequent hypothesis is $a^? \parallel b^*$. At this point the algorithm *stabilizes*: it continues to output $a^? \parallel b^*$ which is

consistent with all examples that can follow. Essentially, this means that the learning algorithm has successfully identified the goal language for this particular *presentation* of the language i.e., the sequence in which examples were presented (assumed to be an exhaustive enumeration). Now, a class of languages is *learnable in the limit* if there is a learning algorithm that identifies any language from the given class with any possible presentation of the language.

Minimal fitting expression. Learning form positive example alone can often be accomplished with an approach that attempts to construct a *minimal fitting expression*, basically a most specific expression that is consistent with the positive examples. Ideally, as it is the case of SORBE_0 and RBE_0 , this expression is unique although it does not need to be the case in general. The class of SORBE_0 is learnable in the limit from positive examples with this method. For instance, take $a^? \parallel b^* \parallel c^+$ as the goal language and consider the execution of the algorithm with the following presentation: the first example is $\{\!|c|\!\}$ and the algorithm returns c^1, the second example is $\{\!|a, c, c|\!\}$ and the output is $a^? \parallel c^+$, the third example is $\{\!|b, c|\!\}$ and the output is $a^? \parallel b^? \parallel c^+$, and finally the third example is $\{\!|b, b, c|\!\}$ and the algorithm stabilizes on the output $a^? \parallel b^* \parallel c^1$. Note that the presentation can be arbitrary and does not need to aim at the quickest stabilization as it would with the presentation of $\{\!|c|\!\}$ followed by $\{\!|a, b, b, c, c|\!\}$.

Infinite elasticity. We point out, however, that the same method does not allow to learn RBE_0 from positive examples. Take for instance the goal expression a^* and the presentation $\{\!|\}\!\}$, $\{\!|a|\!\}$, $\{\!|a, a|\!\}$, $\{\!|a, a, a|\!\}$, etc. At a point i of this sequence, the minimal fitting RBE_0 expression is $a^{[0;i]}$ i.e. the algorithm does not stabilize. In fact, RBE_0 has the property of *infinite elasticity* [Ang80, Wri89], a property that precludes a language from being learnable from positive examples alone. This property, a generalization of *superfinite* languages [Gol78], requires the class of languages to have an infinite sequence of languages strictly included one into each other. Clearly, such a sequence of languages for RBE_0 emerges from the above attempt of learning a^* with the minimal fitting expression technique: $a^{[0;0]} \subseteq a^{[0;1]} \subseteq a^{[0;2]} \subseteq \ldots$ etc.

Learning through enumeration. One of the weak points of the framework of learning in the limit is that it does not impose any limitation on the computational resources used by the learning algorithm. The most prominent manifestation of this issue is that the framework allows learning through a simple *enumeration* of the languages in the given class. Consider for instance a learning algorithm for SORBE_0 from positive and negative examples that enumerates through all SORBE_0 expressions and returns the first expression consistent with the examples seen so far. Because the examples in the presentation are generated by an expression of SORBE_0, the algorithm always find a consistent expression. Furthermore, the algorithm eventually stabilizes because no two SORBE_0 expressions have the same presentation. Note, however, that the number of SORBE_0 expressions is exponential in the number of examples (or more precisely, in the size of the symbols used in the examples), which shows that this kind of approach can be particularly inefficient. We point out that this is not say that SORBE_0 cannot be learned from positive and negative examples more efficiently. In fact, later on, we present two efficient learning algorithms for SORBE_0.

2.3 Learning in polynomial time and data

To address the efficiency concerns, the framework of learning in the limit has undergone a number of modifications [Gol78, dlH97, dlH05]. The main change involves considering learning algorithms that work in an offline manner: a finite set of examples, called a *sample*, is presented on the input and the learning algorithm is required to produce a hypothesis without the knowledge of any prior execution. This allows to measure precisely the computational resources required to

construct a hypothesis, and naturally, the learning algorithm is required to work in polynomial time.

Consistency. The learning algorithm is also required to be *sound* i.e., it needs to output a hypothesis that is consistent with the given input. This requirement alone implies that learning subsumes checking the *consistency* of the input sample i.e., verifying that in fact there exists a language in the given class that is consistent with the input sample. For instance, the sample

$$\{(+, \{\!|a|\!\}), (-, \{\!|a, a|\!\}), (+, \{\!|a, a, a|\!\})\}$$

is not consistent w.r.t. RBE_0 because any RBE_0 accepting $\{\!|a|\!\}$ and $\{\!|a, a, a|\!\}$ must also accept $\{\!|a, a|\!\}$. Consistency is a very useful tool for studying learnability. It allows to use complexity arguments to identify setting where learning is unlikely to feasible. For instance, checking consistency for dRBE_0 is NP-complete [5], and consequently, this class of expressions is unlikely to be learnable from positive and negative examples. On the other hand, a polynomial algorithm for verifying consistency is often a good starting point for a suitable learning algorithm. Take for instance checking consistency of a set of positive and negative examples w.r.t. RBE_0. The approach is relatively simple: we construct the minimal RBE_0 expression fitting the positive examples and verify that it does not satisfy any of the negative examples. The construction of the minimal expression fitting the positive examples consists of identifying the minimum and maximum number of occurrences of every symbol in all positive examples. For instance, the minimal RBE_0 for the positive examples in the following input sample

$$\{(+, \{\!|a, b|\!\}), (+, \{\!|b, b, c|\!\}), (-, \{\!|a, b, b, b, b, c|\!\}), (+, \{\!|a, b, c, c|\!\}), (-, \{\!|a, a, c|\!\})\}$$

is $a^{[0;1]} \parallel b^{[1;2]} \parallel c^{[0;2]}$ and since it is not satisfied by either of the negative examples, the sample is consistent w.r.t. RBE_0. Later on we show how to extend this approach to a full-fledged learning algorithm. We also point out that basically the same approach allows us to test consistency for SORBE_0: the above sample is not consistent w.r.t. SORBE_0 because the minimal SORBE_0 expression fitting the positive examples is $a^? \parallel b^+ \parallel c^*$ and it is satisfied by the negative example $(-, \{\!|a, b, b, b, b, c|\!\})$.

One can argue that the learning algorithm should not be required to verify the consistency of the input sample but rather be allowed to work under the *promise* of a consistent input because in the framework of learning in the limit the presentations are always consistent. We argue that even with the promise, the subsumption of the consistency problem still holds. Namely, a polynomial learning algorithm working under the promise of consistent inputs leads nevertheless to a polynomial procedure for checking consistency. It suffices to time the algorithm running on an input sample, which is consistent if and only if the algorithm stops within the polynomial time and returns an expression consistent with the sample. We point out, however, that consistency checking by a learning algorithm is validated by practical considerations: the user may mislabel examples by mistake, the samples may come from corrupted channels of communications, and also an inconsistent sample may simply indicate that the considered class of languages is not expressive enough to adequately address the needs of the user.

Finally, we point out that when only positive examples are allowed, the consistency problem often becomes trivial, especially if the considered class of languages contains a *universal language* that accepts any set of positive examples. This is the case with RBE_0 and SORBE_0, both containing the universal bag language $a^* \parallel b^* \parallel c^*$.

Characteristic sample. The existence of universal languages clearly shows that polynomial time execution and soundness are alone insufficient to filter out uninteresting and trivial algorithms. For instance an algorithm learning from positive examples returning the universal expression

satisfies the two requirements but can hardly be said to learn anything about the input sample. Furthermore, algorithms that return the minimally fitting expressions are not necessarily suitable either: for RBE_0 this approach never generates an expression with unbounded interval, essentially *overfitting*. Consequently, a *completeness* requirement is added, which is an analogue of the stabilization requirement from the framework of learning in the limit. It requires the algorithm to have the ability to learn any goal language from a sufficiently informative sample. More precisely, a learning algorithm is *complete* if for any goal concept there is a polynomially-sized *characteristic sample* for which the learning algorithm outputs the goal concept. Furthermore, the characteristic sample needs to be *robust* under consistent extensions: the learning algorithm returns the same result even we add to the characteristic sample any number of extra examples (consistent with the goal language).

Learning SORBE_0. Consider a learning algorithm for SORBE_0 from positive and negative examples that constructs the minimal SORBE_0 expression fitting the positive examples and returns it if it does not satisfy any of the negative examples. Naturally, this algorithm is sound and polynomial. For completeness we illustrate the construction of the characteristic sample on the example of $a^1 \parallel b^? \parallel c^*$. The characteristic sample contains two examples that represent the minimum and the maximum number of occurrences of every symbol (note that for c^* it is sufficient to give an example with 2 occurrences of c):

$$\{(+, \{\!|a|\!\}), (+, \{\!|a, b, c, c|\!\})\}.$$

It is easy to see that running the above algorithm with this sample indeed returns $a^1 \parallel b^? \parallel c^*$. Furthermore, the same expression is returned even if further positive examples are added and it is never satisfied by any of the negative examples of the goal expression that can appear on the input. This shows that the algorithm is suitable for learning SORBE_0 from positive examples only as well as from positive and negative examples: when the input sample is consistent, the algorithm does not use negative examples to learn anything.

Formal definition. Because so far we described learning on a rather informal level, we now define it very formally. A *learning setting* for a given class of goal languages $\mathcal{L}$ is a tuple $\mathcal{K} = (\mathcal{R}, \mathit{lang}, \mathit{size}_\mathcal{R}, \mathcal{E}, \mathit{size}_\mathcal{E}, \mathit{examples})$, where $\mathcal{R}$ is a class of representations of $\mathcal{L}$, $\mathit{lang} : \mathcal{R} \to \mathcal{L}$ is a *surjective* map from representations to languages, $\mathit{size}_\mathcal{R} : \mathcal{R} \to \mathbb{N}$ is the size function for representations, $\mathcal{E}$ is the set of examples, $\mathit{size} : \mathcal{E} \to \mathbb{N}$ is the size function for examples, and $\mathit{examples} : \mathcal{L} \to 2^\mathcal{E}$ maps every language to the set of its examples. We extend the example size function $\mathit{size}_\mathcal{E}$ point-wise to finite sets of examples: $\mathit{size}_\mathcal{E}(S) = \sum_{s \in S} \mathit{size}_\mathcal{E}(s)$ for any finite sample $S \subseteq \mathcal{E}$.

Definition (Learnability) Let $\mathcal{K} = (\mathcal{R}, \mathit{lang}, \mathit{size}_\mathcal{R}, \mathcal{E}, \mathit{size}_\mathcal{E}, \mathit{examples})$ be a learning setting for a class of languages $\mathcal{L}$. $\mathcal{L}$ is *learnable in polynomial time and data* in the setting $\mathcal{K}$ iff there exists an algorithm $\mathsf{learner}$ and two polynomials poly_1 and poly_2 such that the following two conditions are satisfied:

1. **Soundness.** For any finite sample $S \subseteq \mathcal{E}$ the algorithm $\mathsf{learner}(S)$ returns a representation $R \in \mathcal{R}$ that is consistent with S i.e., $S \subseteq \mathit{examples}(\mathit{lang}(R))$, or NULL is no such representation exists. The algorithm $\mathsf{learner}(S)$ runs in time $\mathit{poly}_1(\mathit{size}_\mathcal{E}(S))$.
2. **Completeness**. For every language $L \in \mathcal{L}$ and any of representation $R \in \mathcal{R}$ such that $\mathit{lang}(R) = L$ there exists a *characteristic sample* $CS \subseteq \mathit{examples}(L)$ such that $\mathit{size}_\mathcal{E}(CS) \leq \mathit{poly}_2(\mathit{size}_\mathcal{R}(R))$ and for any sample S that extends CS consistently with L i.e., $CS \subseteq S \subseteq \mathit{examples}(L)$, the algorithm $\mathsf{learner}(S)$ returns an equivalent representation $R' \in \mathcal{R}$ of the language L i.e., $\mathit{lang}(R') = L$.

We point out that learnability in polynomial time and data is backward compatible with learnability in the limit. If a class of languages is learnable in polynomial time and data, then the same learning algorithm can be used to show learnability in the limit. Stabilization of the algorithms is a consequence of completeness: eventually the presented examples include the characteristic sample and because of robustness, the learning algorithm returns the goal language from that point on. We also point out that through contraposition, non-learnability in the limit implies non-learnability in polynomial time and data e.g., RBE_0 are not learnable in polynomial time and data from positive examples alone because RBE_0 have infinite elasticity.

2.4 Learning as a search

Now, we illustrate an important concept of search space and show how to explore it using generalizations operations. One of the applications is a learning algorithm for RBE_0 from positive and negative examples. Essentially, the *search space* is the set of all expressions consistent with the positive examples from the input sample. This space is organized by the containment relation that allows to navigate it in search of appropriate solutions. This space may have minimal and maximal elements, which may even be unique, and ideally, it is a lattice. It is the case for RBE_0 and SORBE_0: their search spaces are lattices organized by the containment relation, the bottom element is the minimal expression fitting the positive examples and the top element is the universal expression. As an example in Figure 1 we present the SORBE_0 search space for two positive examples $\{\!|a, a, b, c|\!\}$ and $\{\!|c|\!\}$. We point out that the search space can be large but we do not materialize it.

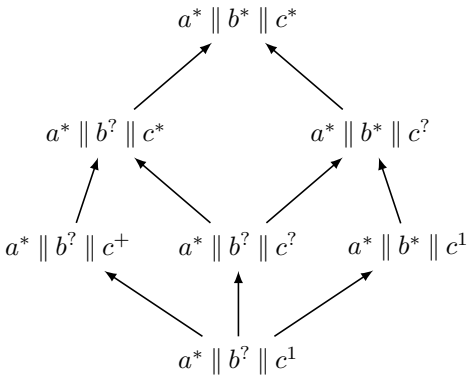

Figure 1 – RBE_0 search lattice for $\{\!|a, a, b, c|\!\}$ and $\{\!|c|\!\}$.

The search space is used for two purposes depending on the direction in which we explore it. Firstly, it can be used to find a minimal fitting expression for a set of positive examples: we start with the universal expression and descend as long as the expression is consistent with the positive examples. This direction corresponds to learning through *specialization*. The second use involves moving in the opposite direction: we typically start at a minimal fitting expression and climb up while making sure that none of the negative examples becomes satisfied. This direction corresponds to learning through *generalization*.

Generalization operations. We organize the search space with the use of generalization operations that closely, if not precisely, capture the containment relation. We observe that the containment of two RBE_0 expressions can be easily characterized with the containment of all

intervals of each pair of corresponding symbols e.g.,

$$a^{[n_a;m_a]} \parallel b^{[n_b;m_b]} \subseteq a^{[n'_a;m'_a]} \parallel b^{[n'_b;m'_b]} \quad \text{iff} \quad [n_a;m_a] \subseteq [n'_a;m'_a] \wedge [n_b;m_b] \subseteq [n'_b;m'_b].$$

This gives rise to a set of RBE_0 generalization operations that work on atoms of the expression and can be expressed with the rewriting rules of the following form

$$x^{[n,m]} \to x^{[n',m']},$$

with $x \in \Sigma$, $n' \leq n$, $m \geq m'$, and $n \neq n'$ or $m \neq m'$. For SORBE_0 the generalization operations are much simpler (with $x \in \Sigma$):

$$x^0 \to x^?, \qquad x^1 \to x^? \qquad x^1 \to x^+ \qquad x^? \to x^*.$$

Generalization operations allow to explore the search space defined by the positive examples in a manner that is *controlled* by negative examples ideally arriving at a desired solution. We illustrate this process on learning algorithms for SORBE_0 and RBE_0.

Learning RBE_0. We use the concept of search space and the generalization operations to build a learning algorithm for RBE_0. First, we illustrate the approach by giving an alternative learning algorithm for SORBE_0.

The alternative learning algorithm for SORBE_0 constructs the minimal SORBE_0 expression fitting the positive examples and then generalize it maximally under the control of negative examples. We illustrate this process on the following input sample:

$$\{(+, \{\!|a, a, b, c|\!\}), (+, \{\!|c|\!\}), (-, \{\!|a, c, c|\!\})\}$$

The search space is the lattice presented in Figure 1 and the initial (minimal fitting) expression is $a^* \parallel b^? \parallel c^1$. This expression can be generalized to $a^* \parallel b^? \parallel c^+$, $a^* \parallel b^? \parallel c^?$, and $a^* \parallel b^* \parallel c^1$. However, that the first expression satisfies the negative example $(-, \{\!|a, c, c|\!\})$ which means that the algorithm needs to choose one of the two latter expressions, say $a^* \parallel b^? \parallel c^?$. Now, this expression can again be generalized in two directions $a^* \parallel b^? \parallel c^*$ and $a^* \parallel b^* \parallel c^?$. The first one satisfies the negative example $(-, \{\!|a, c, c, |\!\})$ hence the generalization process continues in the direction of the second expression $a^* \parallel b^* \parallel c^?$. This expression can further be generalized to the universal expression $a^* \parallel b^* \parallel c^*$ but also this time the negative example prevents this generalization and the expression $a^* \parallel b^* \parallel c^?$ is the final result output by the algorithm.

An analogous approach works for RBE_0 but because for RBE_0 the lattice may be infinite and may have infinite paths of admissible generalization operations, a measure of diligence is required when choosing the right generalization operation: we choose the most maximally generalizing generalization that is admissible by the negative examples. We illustrate the algorithm on the following input sample

$$\{(+, \{\!|a, b|\!\}), (+, \{\!|b, b, c|\!\}), (-, \{\!|a, b, b, b, b, c|\!\}), (+, \{\!|a, b, c, c|\!\}), (-, \{\!|a, a, c|\!\})\}.$$

The minimal fitting RBE_0 expression is $a^{[0;1]} \parallel b^{[1;2]} \parallel c^{[0;2]}$. The atom $a^{[0;1]}$ cannot be generalized because of the negative example $\{\!|a, a, c|\!\}$. The atom $b^{[1;2]}$ cannot be generalized more than to $b^{[0;3]}$ because of the negative example $(-, \{\!|a, b, b, b, b, c|\!\})$. Finally, the atom $c^{[0;2]}$ can be maximally generalized to $c^{[0;\infty]}$. The output of the learning algorithm is $a^{[0;1]} \parallel b^{[0;3]} \parallel c^{[0;\infty]}$.

2.5 Characterizability

We point out the dependence between the algorithm used to learn a class of languages and the construction of the characteristic sample. The characteristic sample that suited the first

algorithm for SORBE_0 does not work for the alternative algorithm. Indeed, for the expression $a^1 \parallel b^? \parallel c^*$ the characteristic sample for the first algorithm is

$$\{(+, \{\!|a|\!\}), (+, \{\!|a, b, c, c|\!\})\}$$

and because it does not have any negative examples the alternative algorithm will generalize the minimal fitting expression with no obstacle to the universal expression $a^* \parallel b^* \parallel c^*$. Because the alternative algorithm depends on the negative examples to control the generalization process, the characteristic sample needs to include negative examples to direct the generalization process away from for all possible overgeneralizations of the goal expression. For $a^1 \| b^? \| c^*$ the overgeneralizations are $a^? \parallel b^? \parallel c^*$ and $a^1 \parallel b^* \parallel c^*$. The corresponding negative examples are $\{\!|b, c, c|\!\}$ and $\{\!|a, b, b, c, c|\!\}$ respectively. All together we get this characteristic sample:

$$\{(+, \{\!|a|\!\}), (+, \{\!|a, b, c, c|\!\}), (-\{\!|b, c, c|\!\}), (-, \{\!|a, b, b, c, c|\!\})\}.$$

The characteristic sample for the learning algorithm for RBE_0 is constructed in an analogous manner.

The dependence of the construction of characteristic sample on the learning algorithm can be considered as *learning bias*: the user might be required to have intimate knowledge of how the learning algorithm works. Robustness of the consistent sample alleviates, to some degree, this concern: the user does not need to know precisely how to construct a characteristic sample but may be required to label more and more examples until it covers a characteristic sample at which point the learning algorithm identifies the goal language. Nevertheless, an interesting question arises: is it possible to propose a *universal characteristic sample* that works for any learning algorithm? In other words, any sound learning algorithm presented with such a sample would return the goal language. The existence of such a sample is independent of any learning algorithm but rather it is a property of a class of languages, know as *finite characterizability* [DP92, dlH97, Dal00]. A class of languages is *finitely characterizable* if for every language in the class there exists a sample that characterizes the language within the class of languages i.e., the language is the sole language consistent with the sample. It is an interesting property because it essentially obsoletes the need of using expression or grammars to specify a language: any language within the given class can be specified with a set of examples. Naturally, we are also interested whether the samples can be relatively small (bounded by a polynomial).

When the class of languages is a lattice whose Hasse diagram is a graph with a bound on the degree of a node, the question of characterizability can be answered positively. The method follows the lines of construction of the characteristic samples for the learning algorithms for RBE_0 and SORBE_0. A language is characterized by a set of positive examples, for which it is the minimal fitting language, and additionally with negative examples obtained from the immediate more general neighbors of the language in the lattice.

2.6 Interactive learning

Good understanding of the structure of the search space also allows to address adequately challenges arising from considering the following framework of *interactive learning*. This framework is particularly suitable for scenarios where the amount of data is too huge for the user to browse and label its elements as positive and negative examples. Instead it is the learning algorithm that identifies the element whose labeling benefits the learning process the most, asks the user to label it, and the process continues until the goal language is reached. This learning framework is closely related to the framework of *Minimum Adequate Teacher* [Ang87], where the learning algorithm asks the user a series of membership queries, *does this element interest you?* and equivalence

queries, *is this the language you have in mind?* The particularity of our considerations is that we wish to minimize the number of interactions. This requires us to quantify the *informativeness* of an unlabeled example in terms of the potential effect that finding out the label (by asking the user).

We illustrate interactive learning on example of SORBE_0 expressions and we assume that the user has already labeled as positive two examples $(+, \{\!|a|\!\})$ and $(+, \{\!|a, b, c, c|\!\})$ and again the minimal fitting expression is $a^* \parallel b^? \parallel c^1$, the bottom element of the search lattice in Figure 1. The bag $\{\!|a, c|\!\}$ is *uninformative* because presenting it for the user to label it does not improve our knowledge about the goal concept. If it is labeled as positive, the minimal fitting expression remains the same (more precisely, the space of consistent expressions remains the same). Labeling it as negative renders the sample inconsistent and since we assume that SORBE_0 contains the goal expression, the example could only be labeled as positive. In fact, one can show that a bag is uninformative if and only if it can be labeled in a way that adding it to the set of previously labeled examples does not render inconsistent.

To understand better which bags are more informative, we need to look closer at the search space, which we consider to capture the state of our knowledge of the goal expression since the goal expression is one of the expressions in the search space. The purpose of further examples is essentially to trim down the space until one element remains. We start at the bottom, the minimal fitting expression, which has 3 outgoing edges, each edge allows to present a relevant bag. For instance, the edge from $a^* \parallel b^? \parallel c^1$ to the more general expression $a^* \parallel b^? \parallel c^?$ yields $\{\!|a, b|\!\}$ which is satisfied by the latter expression but not the former. If the user labels it as positive, then the minimal fitting expression moves up to $a^* \parallel b^? \parallel c^?$, which eliminates from consideration the expressions $a^* \parallel b^? \parallel c^+$ and $a^* \parallel b^* \parallel c^1$. On the other hand, labeling $\{\!|a, b|\!\}$ as negative, eliminates $a^* \parallel b^? \parallel c^?$ and all expressions above it, thus leaving only $a^* \parallel b^? \parallel c^+$ and $a^* \parallel b^* \parallel c^1$ as the only viable candidates.

3 Schemas for unordered and structured data models

Schemas have a number of important functions in databases. They describe the structure of the database, and its knowledge is essential to any user trying to formulate and execute a query or an update over a database instance. Typically, schemas allow for efficient algorithms for validating the conformance of a given database instance. Schemas also capture the intended meaning of the data stored in database instances and are important for static analysis tasks such as query optimization. While relational databases and XML have well-established and widely accepted schema formalisms, the emerging data models of Unordered XML [ABV12] and RDF [AGP09, APR+11] do not have dedicated schema languages. Consequently, we have first proposed suitable a schema language for Unordered XML and studied its basic properties [1] before investigating its learnability [5]. For RDF databases we have developed in collaboration with W3C a suitable schema language [11, 7], which is currently in the process of standardization [W3C13b]. We are currently investigating learning schemas for RDF and report our preliminary findings and ideas.

3.1 Schemas for Unordered XML

When XML is used for *document-centric* applications, the relative order among the elements is typically important e.g., the relative order of paragraphs and chapters in a book. On the other hand, in case of *data-centric* XML applications, the order among the elements may be unimportant [ABV12]. As an example, take a trivialized fragment of an XML document containing the DBLP repository in Figure 2. While the order of the elements `title`, `author`, and `year` may differ from one publication to another, it has no impact on the semantics of the data stored in this semi-structured database (the relative order of authors is often important but it can be explicitly represented with an additional attribute).

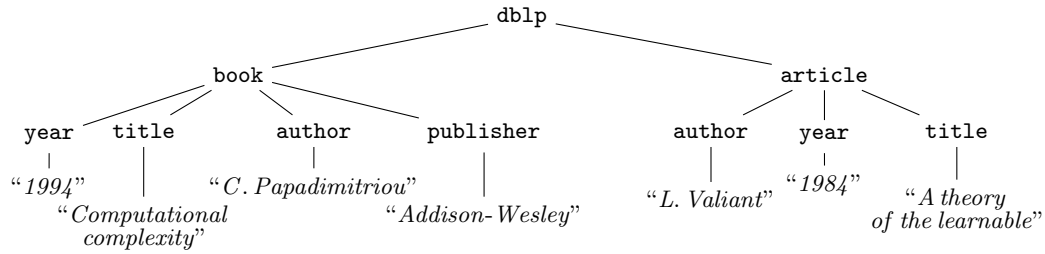

Figure 2 – A trivialized DBLP repository.

Typically, a *schema* for XML defines for every node its *content model* i.e., the children nodes it must, may, and cannot contain. For instance, in the DBLP example, one would require every article to have exactly one title, one year, and one or more authors. A book may additionally contain one publisher and may also have one or more editors instead of authors. The *Document Type Definition* (DTD), the most widespread XML schema formalism for (ordered) XML [BNVdB04, GM11], is essentially a set of rules associating with each label a regular expression that defines the admissible sequences of children. The DTDs are best fitted for ordered content because they use regular expressions, a formalism that defines sequences of labels. However, when unordered content model needs to be defined, there is a tendency to use *over-permissive* regular expressions. For instance,

the DTD below corresponds to the one used in practice for the DBLP repository [1]:

$$\begin{aligned} \texttt{dblp} &\rightarrow (\texttt{article} \mid \texttt{book})^* \\ \texttt{article} &\rightarrow (\texttt{title} \mid \texttt{year} \mid \texttt{author})^* \\ \texttt{book} &\rightarrow (\texttt{title} \mid \texttt{year} \mid \texttt{author} \mid \texttt{editor} \mid \texttt{publisher})^* \end{aligned}$$

This DTD allows an article to contain any number of titles, years, and authors. A book may also have any number of titles, years, authors, editors, and publishers. These regular expressions are clearly over-permissive because they allow XML documents that do not follow the intuitive guidelines set out earlier e.g., an XML document containing an article with two titles and no author should not be valid.

While it is possible to capture unordered content models with regular expressions, a simple pumping argument shows that their size may need to be exponential in the number of possible labels of the children. This suggests that over-permissive regular expressions may be employed for the reasons of conciseness and readability, a consideration of great practical importance.

We propose Unordered XML Schemas (UXS), a model of schemas analogous to DTDs that instead of using regular expressions defining sequences of admissible children nodes, uses regular bag expressions (RBE) defining admissible collections of children nodes. For instance, take the following schema (satisfied by the tree in Figure 2):

$$\begin{aligned} \texttt{dblp} &\rightarrow \texttt{article}^* \parallel \texttt{book}^* \\ \texttt{article} &\rightarrow \texttt{title} \parallel \texttt{year} \parallel \texttt{author}^+ \\ \texttt{book} &\rightarrow \texttt{title} \parallel \texttt{year} \parallel \texttt{publisher}^? \parallel (\texttt{author}^+ \mid \texttt{editor}^+) \end{aligned}$$

The above schema captures the intuitive requirements for the DBLP repository. In particular, an article must have exactly one title, exactly one year, and at least one author. A book may additionally have a publisher and may have one or more editors instead of authors. Note that, unlike the DTD defined earlier, this schema does not allow documents having an article with several titles or without any author.

3.2 Regular bag expressions

A *regular bag expression* (RBE) defines bags by using disjunction "|", unordered concatenation "‖", and unordered Kleene star "$*$". For example, $(a \parallel b)^* \parallel (c^* \mid d^*)$ defines a set of bags that have the same number of a's and b's and either an arbitrary number of c's or an arbitrary number of d's. Additionally, the option operator ? can be defined with disjunction e.g., $(a \parallel b^?)^*$ defines a set of bags that have no more b's than a's. An expression is *single-occurrence* (SORBE) if every symbol it uses is used at most once. A number of important facts are known about RBEs: they are closed under intersection, union, and complement [Opp78]. Also, when a bag of symbols is viewed as vector of natural numbers (obtained by fixing some total order on the set of symbols), RBEs are equivalent to the class of semilinear sets and the class of vectors definable with Presburger arithmetic [GH66, Par66].

A number of complexity results has also been previously established: testing *membership* i.e., that a given bag satisfies a given RBE, is NP-complete [KT10] and so is testing the *emptiness of intersection* of two RBEs [CGS09]. We have, however, improved our understanding of the complexity of RBE. We have shown that membership remains NP-complete even if the expression uses only the unordered concatenation ‖ and the option ? operator while allowing symbol repetitions. Also, we have shown that testing containment between two RBEs is Π_2^p-hard and in

1. `http://dblp.uni-trier.de/xml/dblp.dtd`, Retrieved on 3 Nov 2015.

3EXPTIME. Consequently, we have identified a number of subclasses with desirable computational properties. We have shown that membership for SORBE is in PTIME even if we allow the use of intervals on symbols e.g., $(a^{[2;\infty]} \parallel b^{[3;4]})^*$. We have also identified a class of *disjunctive interval multiplicity expressions* (DIMEs) with tractable containment problem. DIME is a rich subclass of SORBE with intervals and allowing a degree of alternation of disjunction and unordered concatenation.

For the purposes of learning schemas we have studied two proper subclasses of DIME: SORBE_0 consisting of unordered concatenations of atoms of the form a^M with $M \in \{0, 1, ?, +, *\}$, and dSORBE_0 consisting of concatenations of disjunction of such atoms. For instance, the expression $\texttt{title} \parallel \texttt{year} \parallel \texttt{author}^+$ is SORBE_0 while the expression $\texttt{title} \parallel \texttt{year} \parallel \texttt{publisher}^? \parallel (\texttt{author}^+ \mid \texttt{editor}^+)$ is dSORBE_0.

3.3 Complexity of Unordered XML schemas

We have investigated the basic computational properties of two classes of unordered schemas depending on the class of bag expressions used in the rules: UXS_0 allows using only SORBE_0 expressions and dUXS_0 only dSORBE_0 expressions. We have studied the complexity of the standard decision problems involving schemas: *Membership* – does a given document satisfies the given schema?; *Schema containment* – does every document satisfying one schema satisfies the other schema?; *Query satisfiability* – is the given twig query satisfied in some document specified the schema?; *Query implication* – is the given twig query satisfied in every document specified by the schema?; and *Query containment* – does satisfaction of one query implies satisfaction of another query over the documents specified by the schema? The results are summarized in Table 1. We point out that there is a close resemblance between the complexity results for UXS and those

Problem of interest	dUXS_0	UXS_0
Membership	PTIME	PTIME
Schema containment	PTIME	PTIME
Query satisfiability	NP-complete	PTIME
Query implication	EXPTIME-complete	PTIME
Query containment	EXPTIME-complete	coNP-complete

Table 1 – Summary of complexity results for classes of Unordered XML Schemas.

for (general and disjunction-free) DTDs [BKW98, Sch04, MNS09, BFG08, NS06]. However, many of our results are novel because the results for DTDs often rely on the order expressed with regular expressions and could not be adapted to the unordered setting. Consequently, we have developed novel approaches of analyzing the complexity in the unordered setting.

3.4 Learning Unordered XML schemas

The natural way to learn DTD-like schema is by *bootstrapping* the learning algorithm for the expressions that specify admissible collections of children nodes [HNW06, GGR$^+$03, BNST06, BNSV10, FK13], and this is also the approach we adopt for learning schemas for Unordered XML. A positive example for learning UXS is an unordered tree. For given a label, every node with the label generates one positive bag example, consisting of the labels of the children of the node, for learning the bag expression to be used as the definition of the contents of the given label. Since SORBE_0 is learnable from positive examples, so is UXS_0. The learning algorithm generates the unique minimal UXS_0 schema fitting the positive examples.

Learnability of dUXS_0 follows from an alternative characterization of dSORBE_0 in the form of a graph that essentially identifies mutually exclusive pairs of symbols (in fact, it works for the larger class of DIMEs). This characterization allows to test efficiently containment but also allows to identify minimal dSORBE_0 expression fitting a given set of positive examples. However, there might be more than one such expression, in fact an exponential number of minimal fitting dSORBE_0 expressions. This explosion has negative computational consequences: testing consistency of dSORBE_0 is intractable, which precludes feasible learnability of dUXS_0 from positive and negative examples.

We point out that while a positive example for learning UXS is very informative as it generates a number of positive examples for learning relevant bag expressions, a negative example does not provide much information. A negative example is tree that is not valid w.r.t. goal schema but typically we do not know why it is not valid i.e., which nodes invalidate the schema. Consequently, it is unlikely to generate negative examples for learning the bag expressions. The learning algorithm for UXS_0 from positive and negative examples is the same algorithm as for learning UXS_0 from positive examples alone with the additional consistency. Table 2 contains the summary of learnability results for Unordered XML Schemas.

Schema formalism	*+ examples only*	*+ and - examples*
UXS_0	Yes	Yes
dUXS_0	Yes	No

Table 2 – Summary of learnability results for classes of Unordered XML Schemas.

3.5 Shape Expression Schemas for RDF

While RDF has originally been proposed as schema-free data format, recently the need for a schema language has been clearly identified [W3C13a], the language of Shape Expressions Schemas (ShEx) has been proposed [11, 7], and it is currently under development by W3C [W3C13b]. ShEx builds on the success of XML Schema and allows to define a set of node types that use regular bag expressions to constraint the immediate (outbound) neighborhoods of the nodes. Take for instance the RDF graph in Figure 3 storing bug reports. And consider this the following intuitive

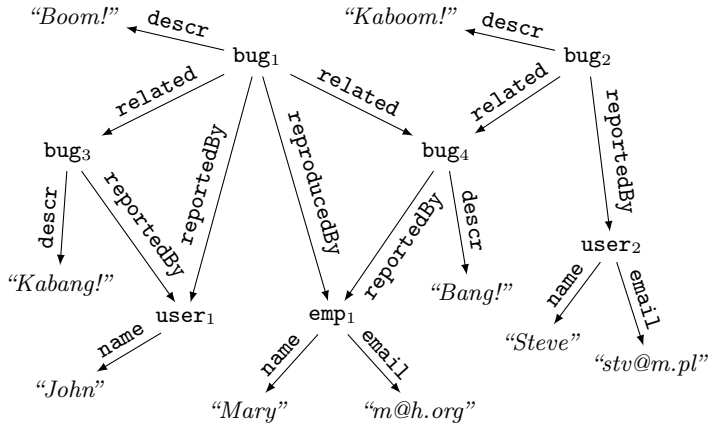

Figure 3 – An RDF graph with bug reports.

ShEx schema:

$$\mathsf{Bug} \to \texttt{descr} :: \mathsf{string} \parallel \texttt{reportedBy} :: \mathsf{User} \parallel \texttt{reproducedBy} :: \mathsf{Employee}^{?} \parallel \texttt{related} :: \mathsf{Bug}^{*}$$

$$\mathsf{User} \to \texttt{name} :: \mathsf{string} \parallel \texttt{email} :: \mathsf{string}^{?}$$

$$\mathsf{Employee} \to \texttt{name} :: \mathsf{string} \parallel \texttt{email} :: \mathsf{string}$$

The schema says that a bug report has a description and a user who reported it. Optionally, a bug report may also have an employee who successfully reproduced the bug. Finally, a bug report can have a number of related bug reports. A user has a name and an optional email address while an employee has a name and a mandatory email address. ShEx has the natural semantics: the input graph is valid if it is possible to assign types to nodes in a manner that satisfies all shape expressions of the schema.

We have actively contributed to the effort of W3C and have studied [11, 7] the semantics and the complexity of validation for ShEx. We have considered two semantics depending on whether or not a node can have more than one type. The *single-type* semantics requires every node to have precisely one type while the *multi-type* semantics allows nodes to have multiple nodes. Take for instance the graph and the schema in in Figure 3 and note that while the node $\texttt{user}_2$ satisfies the types Employee and User, the typing of the RDF graph needs to assign to this node only one type User. However, the node $\texttt{emp}_1$ needs to have both the type Employee and the type User, and consequently, the graph is valid only according to the multi-type semantics.

The single-type semantics makes the validation problem a generalization of graph homomorphism, and consequently, single-type validation is NP-complete even for very simple schemas. On the other hand, the multi-type semantics is closer to graph simulation, which makes it computationally more attractive. In fact, we have established a close relationship between multi-type validation and testing emptiness of the intersection RBE expressions. While testing emptiness of intersection for RBEs is known to be intractable [CGS09], restricting the class of expression use in schema to RBE_0 renders validation tractable. Furthermore, we have proposed a natural condition of *determinism* for ShEx: given a type of a node and the label of an outgoing edge, the type that the target node has to satisfy is unique. The schema in Figure 3 is deterministic but replacing the rule for Bug with the rule

$$\mathsf{Bug} \to \texttt{descr} :: \mathsf{string} \parallel \texttt{reportedBy} :: \mathsf{User} \parallel (\texttt{reproducedBy} :: \mathsf{Employee}^{?} \mid \texttt{reproducedBy} :: \mathsf{User}^{?}) \parallel \texttt{related} :: \mathsf{Bug}^{*}$$

renders the schema nondeterministic because there are two types that need to be considered when following an edge with the label reproducedBy. Interestingly, for deterministic schemas the problem of validation is reduced to the problem of membership for bag expressions. While it is also intractable [KT10], it is simpler than testing emptiness of intersection of RBEs and permits tractable validation for a broader class of single-occurrence regular bag expressions (SORBE).

3.6 Learning Shape Expression Schemas

Learnability of Shape Expression Schemas is largely an open question, and we report on our preliminary findings and ideas on how to approach it.

We restrict our attention to ShEx_0 the subclass of Shape Expression Schemas that uses only SORBE_0 expressions. This restriction has the advantage of an alternative equivalent representation in the form of *shape graphs*, where nodes are types and edges are additionally labeled the occurrence constraint in $\{0, 1, ?, +, *\}$. For instance, Figure 4 contains the shape graph corresponding to the schema for the RDF graph from Figure 3. The validity of a graph w.r.t. the a ShEx_0 schema

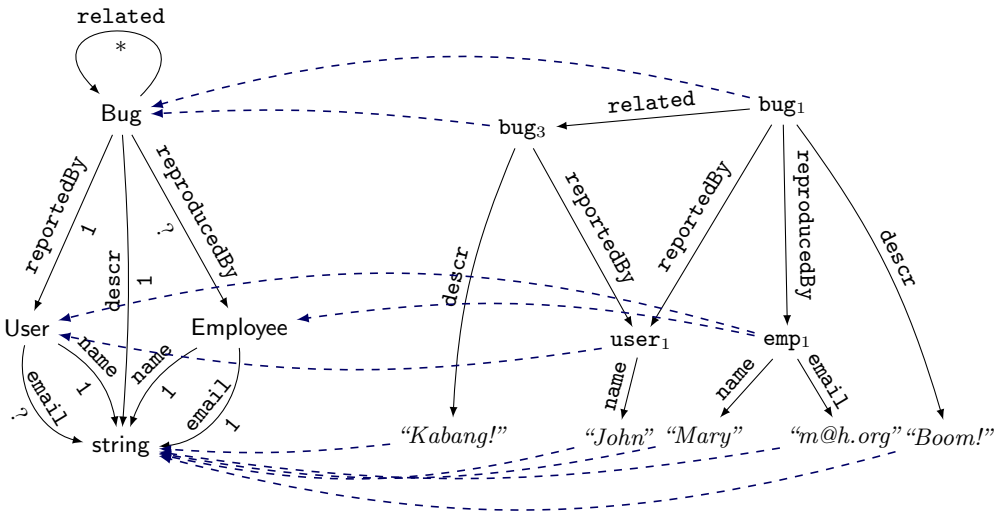

Figure 4 – Embedding of an RDF graph into a shape graph.

is defined with a natural notion of embedding (cf. Figure 4). Interestingly, any RDF graph is also a shape graph, and in fact, it is the minimal fitting shape graph. Furthermore, the notion of embedding can be extended to pairs of shape graphs, and it is a sufficient condition for containment of two shape graphs although it is not necessary a condition. While the exact complexity of the containment problem for ShEx_0 remains an open problem and we suspect it to be intractable, we have shown that testing embedding between two ShEx_0 is in PTIME. Consequently, we investigate using the embedding to define the search (sub)space and explore it using generalization expressions that can be divided into two groups: 1) occurrence constraint relaxation $1 \to ?$, $1 \to +$, $? \to *$, and $+ \to *$, 2) fusing two types.

We illustrate the use of the generalization operations and the embeddings by outlining two prospective approaches of learning ShEx_0 that we currently investigate. The first approach attempts to construct a deterministic ShEx_0 by iteratively fusing any sets of types that are reachable from another type with an edge of the same label (note that when we treat an RDF graph the notions of a type and node coincide). For instance, for the RDF graph in Figure 3, the node $\mathtt{bug}_2$ yields the following rule

$$\mathtt{bug}_2 \to \mathtt{descr} :: \mathsf{string} \parallel \mathtt{reportedBy} :: \mathtt{user}_2 \parallel \mathtt{related} :: \mathtt{bug}_4$$

However, generating the rule for $\mathtt{bug}_1$ requires fusing the types $\mathtt{bug}_3$ and $\mathtt{bug}_4$, and by extension also the types $\mathtt{user}_1$ and $\mathtt{emp}_1$.

$$\mathtt{bug}_1 \to \mathtt{descr} :: \mathsf{string} \parallel \mathtt{reportedBy} :: \mathtt{user}_1 \parallel \mathtt{reproducedBy} :: \mathtt{emp}_1 \parallel \mathtt{related} :: \{\mathtt{bug}_3, \mathtt{bug}_4\}^+$$
$$\{\mathtt{bug}_3, \mathtt{bug}_4\} \to \mathtt{descr} :: \mathsf{string} \parallel \mathtt{reportedBy} :: \{\mathtt{user}_1, \mathtt{emp}_1\}$$
$$\{\mathtt{user}_1, \mathtt{emp}_1\} \to \mathtt{name} :: \mathsf{string} \parallel \mathtt{email} :: \mathsf{string}^?$$

With a combinatorial argument we have shown that this procedure always terminates and introduces at most quadratic number of new types. The newly introduced types are more general than their component types and with the use of embeddings we can *reduce* the schema by canonizing types (fusing together types know to be equivalent) and diligently eliminating types

that are subsumed by other types. For instance, the type $\texttt{bug}_2$ is subsumed by the type $\texttt{bug}_1$ and can be removed from the schema without changing its semantics.

The second approach works in a three phases (and does not use determinism). In the first phase, we take the input graph G and add $*$ on every of its edges to obtain a shape graph $G^\star$ with $*$ only. In the second phase we take the (maximal) autoembedding of $G^\star$ and construct the *subtype reduct* $G^\prec$ by replacing in $G^\star$ every type with a maximally more general type indicated by the autoembedding. In the third final stage, we take an embedding of G into $G^\prec$ and use it to specialize the occurrence constraints of $G^\prec$ obtaining this way the final shape graph G°. This process is illustrated in Figure 5 on the graph from Figure 3. We note that in $G^\star$ the type $\texttt{bug}_2$

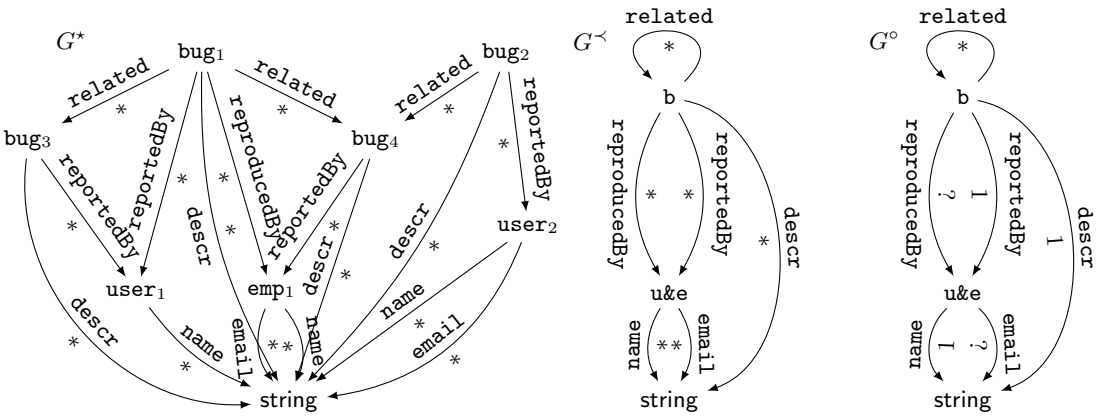

Figure 5 – Learning ShEx_0: generalization (left), reduction (middle), and specialization (right)

is embedded to type $\texttt{bug}_1$, which creates the self loop of the type `b` in $G^\prec$. Also, in $G^\star$ the types $\texttt{emp}_1$ and $\texttt{user}_2$ are equivalent and subsume the type $\texttt{user}_1$, all which are fused to the type `u&e` in $G^\prec$. Finally, the embedding of G (Figure 3) assigns the type `u&e` to nodes that have exactly one outgoing edge labeled with `name` and at most one outgoing edge labeled with `email`, which specializes the cardinality constraints on those edges in G° to 1 and ? respectively.

We point out that if H is a shape graph (goal schema) that is satisfied by G i.e., G can be embedded in H, then H can be embedded in $G^\prec$. This means that G and $G^\prec$ are two antipodes of a search space, organized with the embedding relation, that contains all schemas of interest. The second learning algorithm described above explores this search space but only partially: only elements obtained by specialization of occurrence constraints of $G^\prec$ can be reached. In the example in Figure 5 this behavior manifests itself by lack of two distinct types for employees and users. One of the reasons is that (in the schema from the running example) the type User is a subtype of the type Employee. Currently, it is not clear if positive examples alone are sufficient to distinguish the two types by a learning algorithm.

4 Twig queries

In this section we present on our work on learning and characterizing twig queries [13, 14]. The class of *twig queries* [AYCLS02] is a practical subclass of XPath [W3C99, W3C07a] that captures the core querying functionality of any XML database and is the basis of more advanced query languages such as XQuery [W3C07b]. Twig queries allow to query XML documents using a syntax similar to directory paths used to navigate in the UNIX file system, with $\star$ matching any element and // allowing to access all descendants of the current node. Twig queries can also be presented as *tree patterns* and their semantics is defined with a natural notion of embedding. Figure 6 present an example of an XML library catalog and a unary twig query that selects titles of works authored by Karol Marx, written in XPath syntax as `/library/⋆[author="K. Marx"]/title`.

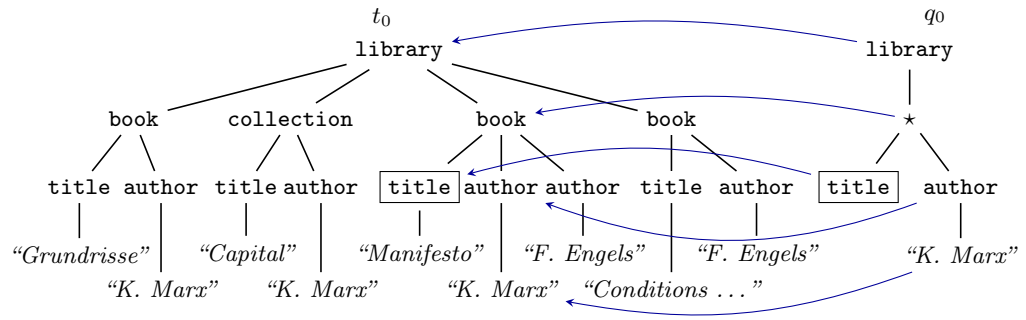

Figure 6 – Embedding of a unary twig query q_0 into a tree t_0.

While in general twig queries can be of arbitrary arity i.e., select tuples of nodes, it is typically sufficient to study their Boolean variant and the results carry over to n-ary queries [MS04]. Naturally, Boolean queries have their uses, for instance Figure 7 presents an example of filtering a simple XML feed with offers from a consumer-to-consumer web site with the use of a Boolean twig query selecting offers for sale, written in XPath syntax `.[offer//item/type="For sale"]`.

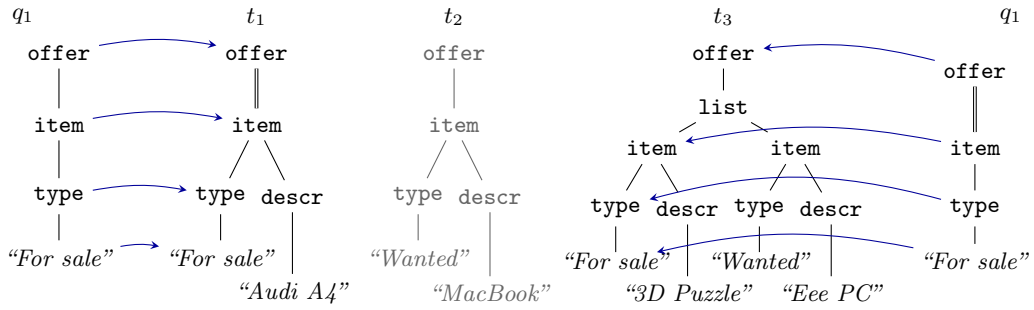

Figure 7 – Filtering a stream of XML documents $(t_1, t_2, t_3, \ldots)$ with Boolean twig query q_1.

4.1 Fundamental obstacles

Learning twig queries from positive and negative examples is unlikely to be feasible as we have shown the consistency problem for twig queries to be NP-complete. Consequently, we focus on learning twig queries from positive examples only, and later on we outline a possible approach

to learning unions of twig queries for which the consistency problem is trivial. The learnability of the full class of twig queries from positive examples alone remains an open question but some evidence suggests that it is unlikely to be feasible. Firstly, the containment of twig queries is know to be coNP-complete [MS04]. Furthermore, we have shown in [14] that characterizing twig queries may require exponentially many examples.

We pursue an approach to learn twig queries by attempting construct the minimal query fitting the input. As we illustrate in Figure 8, the minimal twig query fitting a set of positive examples may be of size exponential in the size of the input, and consequently, constructing precisely the minimal fitting query would not be an efficient approach. Instead we construct an

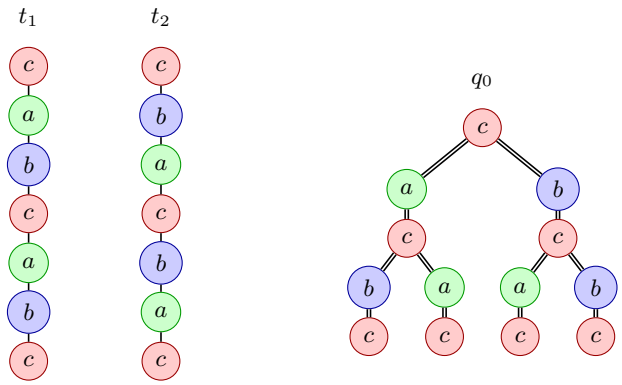

Figure 8 – Minimal twig query q_0 fitting t_1 and t_2.

approximation of the minimal fitting query (which for the examples in Figure 8 outputs a query that is a single path of the exponential minimal fitting query).

4.2 Learning twig queries from positive examples

Our learning algorithm is inspired by algorithms for inference of word patterns [Ang79, Shi82] (see [SA95] for a survey of the area). We illustrate our learning algorithm on an example of a unary query that selects library items written by Karol Marx, with two positive examples presented in Figure 9. The algorithm essentially works in two phases. In the first phase the

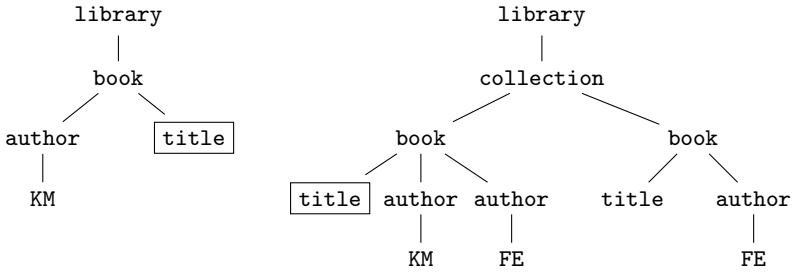

Figure 9 – Two positive examples.

algorithms infers the selecting path: it begins with the universal query //⋆ that selects all nodes and iteratively specializes it. First by identifying common elements on every path:

//⋆ → //library//⋆ → //library//book//⋆ → //library//book//title

Next, it specializes the descendant edges by replacing them with child edges if possible:

`//library//book//title` $\rightarrow$ `/library//book//title` $\rightarrow$ `/library//book/title`

In the second phase, it identifies most specific (Boolean) path queries that are common to both examples and *weaves* it into the unary path query. In the examples in Figure 9, there is one such path query: `/library//book/author/KM`. And the result of weaving it into the unary path query is

`/library//book[author/KM]/title`.

This query is also the output of the learning algorithm.

4.3 Anchored twig queries

The described algorithm works for a practical subclass of *anchored* twig queries. * A twig query is *anchored* (ATwig) if no descendant edge is incident to a $\star$ node unless this node is a leaf node. A number of anchored and non-anchored queries is presented in Figure 10. In essence,

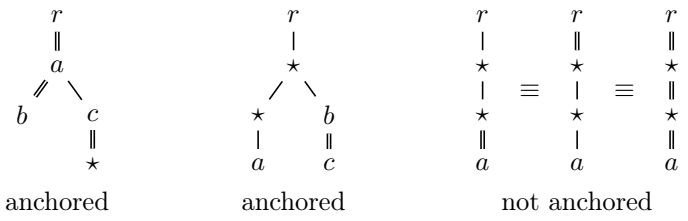

Figure 10 – Anchored and non-anchored twig queries.

anchored twig queries do not allow cannot express conditions on ancestry of nodes with a minimal distance between them. For instance, the non-anchored queries in Figure 10 check the existence of a a labeled node at the depth at least 2.

The containment of anchored twig queries is in PTIME because it is equivalent to the existence of an embedding. We point out that for the arbitrary twig queries the existence of an embedding is a sufficient but not a necessary condition for containment. The equivalence of containment and embeddings, a structural characterization of containment of anchored twig queries, allows us to capture the search space with a set of generalization operations.

4.4 Generalization operations and normalization

We employ simple generalization operations, presented in Figure 11, where dashed lines indicate optional arbitrary edges, x, y and z may have arbitrary labels, including $\star$, while a ranges over non-$\star$ symbols only. The generalization operations allow to (δ_1) change a non-$\star$ label to $\star$, (δ_2) change a child edge to a descendant edge, and (δ_3) remove a node while connecting all its children to the parent node with descendant edges. We point out that applying a generalization operation to an anchored twig query may result in a query that is not anchored. We allow applying a sequence of generalization operations to an anchored query q that lead to an anchored twig query p only if there is no anchored middle point between the two queries (on any sequence of generalization operations leading from q to p). An essential result here states that the number

*. In the original article [13] an additional restriction to *path-subsumption-free* queries has been imposed. We have since significantly weakened this restriction and currently work on removing it all together.

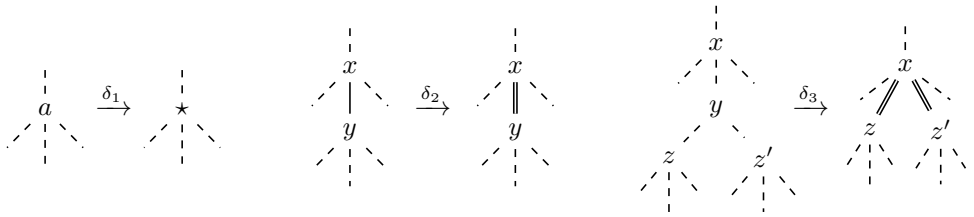

Figure 11 – Generalization operations ($\rightarrow$).

of p's that can be reached in this way is polynomial in the size of q, and in fact, they can be defined with a small tractable set of macro-operations.

An immediate application of the generalization operations is *normalization* of anchored twig queries. Basically, we apply a generalization operations as long as doing so yields an equivalent query and with this process we obtain the size-minimal equivalent query. This process is tractable because the number of possible generalizations to consider is polynomial. Figure 12 presents an example of normalizing an anchored twig query and illustrates the close connection between the range of an autoembedding and the outcome of the normalization. In fact, an anchored twig query is normalized if and only if its only autoembedding is an identify function.

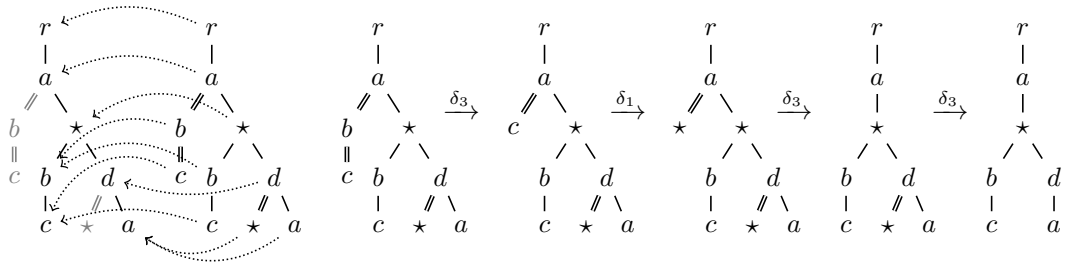

Figure 12 – Normalizing a query.

4.5 Characterizability

To approach the problem of characterizability we explore further the connection between embeddings and generalization operations and show that one anchored twig query can be generalized into another anchored twig query if and only if there exits an *injective* embedding between the queries. This is illustrated with the embedding between q_1 and q_2 in Figure 13. We point out that a number of types injective embeddings has been studied in the literature, which we have surveyed in [10]. For our purposes, we employ *ancestor-preserving* embeddings and we point out that while ancestor-preserving embedding are injective functions not every embedding that is an injective function is ancestor preserving. For instance, in Figure 13 the embedding of query q_0 in t_0 is ancestor-preserving but the embedding of q_0 in t_1 is not despite being an injective function.

If instead of using the standard notion of embeddings we use the injective embedding to define the semantics of anchored queries, then the equivalence between ancestor-preserving embeddings and generalization operations allows to map very precisely the lattice of anchored twig queries. More precisely, we can use the generalization operations to identify the neighborhood of an

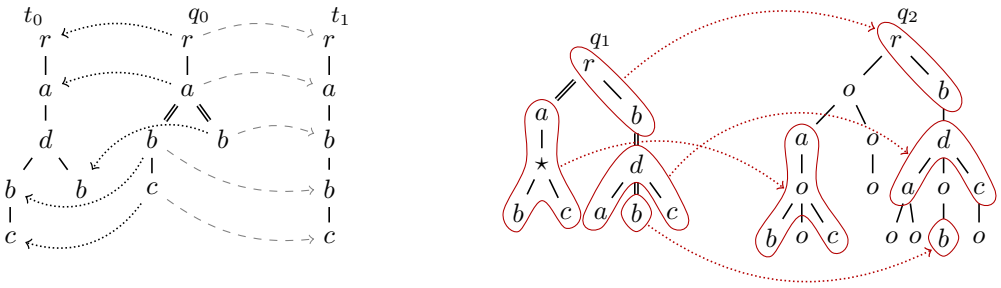

Figure 13 – Ancestor-preserving injective embeddings.

anchored twig query q in the lattice and to show that it is of polynomial size. For instance, in Figure 14, the immediate generalizations of p_0 are p_0', p_0'', and p_0'''. This neighborhood can then be used to generate negative examples and together with positive examples for q gives a polynomially sized set of examples that characterizes q under injective semantics.

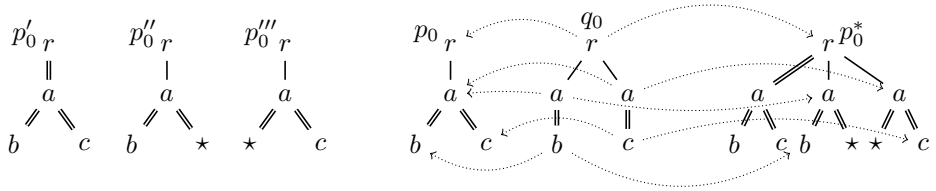

Figure 14 – Immediate anchored generalizations versus an embedding with overlap.

We point out that this approach does not work for the standard semantics which can embed one query into another with *overlap*. This is illustrated in Figure 14: the query q_0 is more general than p_0 and yet it cannot be embedded into any of the immediate generalization of p_0. To handle the overlap problem we diligently apply a duplication operation that creates separate copies of a fragment of the query and apply different generalization operations on each copy. This is illustrated in Figure 15 and also in Figure 14 where p_0^* is obtained from p_0 in such a manner. As

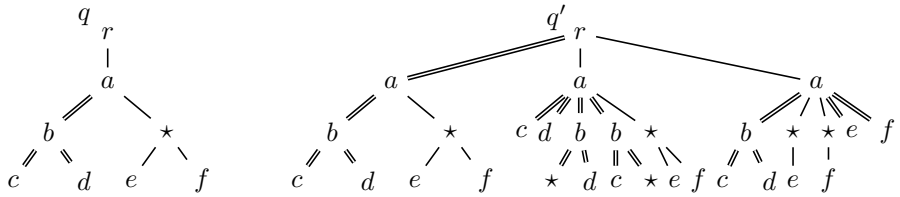

Figure 15 – Constructing the minimal generalization q' of a query q.

a result anchored twig queries are characterizable with polynomially-sized sets of positive and negative examples. Finally, we add that we have shown the full class of twig queries to be also finitely characterizable but there exist queries that require exponential sets of examples. We have also shown that unions of twig queries are not finitely characterizable.

4.6 Learning unions of twig queries

One of the reasons of the high complexity of consistency check for twig queries, which precludes learning twig from positive and negative examples, is that the class of twig queries is so constrained that the consistency check is reduced to solving constraints. Consequently, considering a richer class of queries can possibly lead to learnability and we consider unions of twig queries.

We bring our attention to unions of twig queries for which the consistency problem becomes trivial. A tree is also a twig query, and in fact, it is the most specific twig query that the tree satisfies. Consequently, for a given set of positive and negative examples, if we consider its positive examples as twig queries and take their union, then this union query is the unique minimum fitting query and the sample is consistent if and only if none of the negative examples is satisfied by the union. For instance, if we take the examples in Figure 16, the minimal (Boolean) union

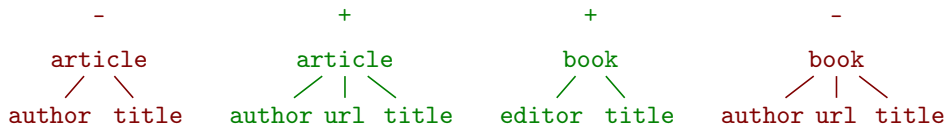

Figure 16 – A set of positive and negative examples.

query fitting the positive examples is .[article[author][url][title]] ∪ .[book[editor][title]].

The learnability of unions of twig queries remains an open question and currently we investigate learnability of *unions of anchored twig queries* (UATwig). We are encouraged by the fact that when restricted to anchored twigs, unions have a simple characterization of containment: P is contained in Q if and only if for every twig component p of P there is a twig component q of Q such that p can be embedded in q. The algorithm that we investigate begins with the minimal fitting query and iteratively applies generalization operations to its components under the control of negative examples (i.e., making sure that none of the negative examples are satisfied). In Figure 17 we present its execution on the sample from Figure 16. The end result is

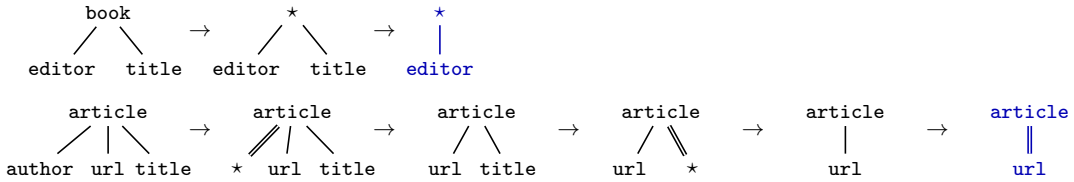

Figure 17 – Learning unions of anchored twig queries from positive and negative examples.

.[⋆/editor] ∪ .[article//url].

Query class	*+ examples only*	*+ and − examples*
ATwig	Yes *	No
UATwig	No	Open

Table 3 – Learnability of anchored twig queries.

We point out that the results on characterizability of anchored twig queries allow us to prove that this algorithm is not only sound but also almost complete: for any query we can construct

a sample for which the learning algorithm returns the given query, but extending this sample with additional examples may cause the algorithm to construct a different query. It remain to be seen if the algorithm can be altered to handle properly robust characteristic samples. Finally, we observe that the class of unions of twig queries has the property of infinite elasticity, and consequently, they are not learnable from positive examples alone. Table 3 contains a summary of learnability results for twig queries.

5 Join queries

In this section we present our results on learning semi-join and equi-join queries for relational databases in an interactive scenario. In the interactive scenario the algorithm presents examples for the user to label them as positive or negative [3, 4] and the main objective is to minimize the number of examples that the user needs to label before the algorithm is able to identify the goal query. This work has lead to a general paradigm for learning queries on Big Data [2].

While the class of join queries that we consider is relatively simple, it captures two fundamental query mechanisms: *equi-join* queries identify matching pairs of records and *semi-join* queries filter records in one table based on the contents of another. We illustrate the use of join queries on the example of a user who wants to build a list of flight&hotel packages. The users has access

Flights

From	*To*	*Airline*
Paris	Lille	AF
Lille	NYC	AA
NYC	Paris	AA
Paris	NYC	AF

Hotels

City	*Discount*
NYC	AA
Paris	None
Lille	AF

Figure 18 – A relational database with flights and hotels.

to two tables *Flights* and *Hotels* with an example of their contents presented in Figure 18: the *Discount* attribute in *Hotels* indicates a discounted price if the hotel reservation is made together with a flight with the indicated airline. A complete list of all possible packages, including those without a hotel discount, is obtained with this equi-join query

$$Q_1 = Flight \bowtie_{Flight.To=Hotel.City} Hotel$$

while the list of packages with discounted hotel rates is obtained with the equi-join query

$$Q_2 = Flight \bowtie_{Flight.To=Hotel.City \wedge Flight.Airline=Hotel.Discount} Hotel.$$

The following example of a semi-join query selects flights to cities where it is possible to stay in a hotel with discounted price (without listing the hotels)

$$Flight \ltimes_{Flight.To=Hotel.City \wedge Flight.Airline=Hotel.Discount} Hotel.$$

We study classes of join queries of an arbitrary number of tables whose join conditions are restricted to either conjunctions or disjunction of conjunctions of equality tests between attributes of two different tables. Because the type of the join query, equi-join or semi-join, is known from the context, we represent queries with (possibly nested) subsets of pairs of predicates. For instance, Q_2 is represented with $\{(To, City), (Airline, Discount)\}$ while disjunctions introduce a level of nesting e.g., $R \bowtie_{(R.A=P.B \wedge R.C=P.D) \vee R.A=P.D} P$ is represented as $\{\{(A, B), (C, D)\}, \{(A, D)\}\}$. We point out that this representation allows to define the lattice of all join queries with very simple generalization operations, which basically remove equality tests from the join conditions. For instance, Figure 19 presents the lattice of join queries using conjunctions of equalities for the relational database in Figure 18.

5.1 Learning join queries

We study learnability of join queries in the setting where the number and the names of the tables are known. We point out that the output of an equi-join query is a subset of the Cartesian

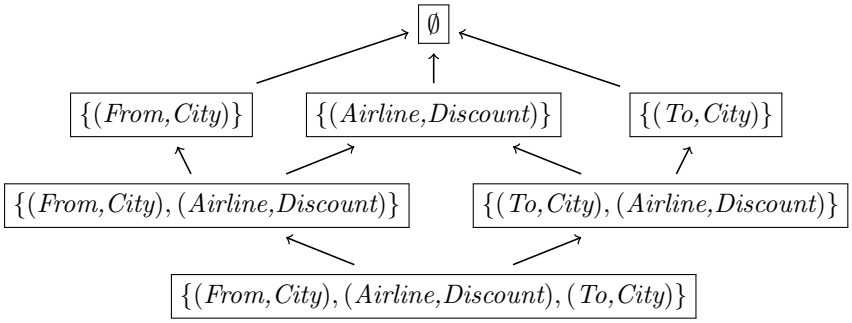

Figure 19 – Lattice of join predicates for the product table in Figure 20.

product of the tables. Consequently, we consider a scenario, where the user labels as positive and negative tuples in the product table. For instance, Figure 20 contains the product table of the *Flight* and *Hotel* tables with a number of labeled tuples. Because we are interested in learning

	From	*To*	*Airline*	*City*	*Discount*	
	Paris	Lille	AF	NYC	AA	(1)
	Paris	Lille	AF	Paris	None	(2)
+	Paris	Lille	AF	Lille	AF	(3)
+	Lille	NYC	AA	NYC	AA	(4)
	Lille	NYC	AA	Paris	None	(5)
	Lille	NYC	AA	Lille	AF	(6)
	NYC	Paris	AA	NYC	AA	(7)
−	NYC	Paris	AA	Paris	None	(8)
	NYC	Paris	AA	Lille	AF	(9)
	Paris	NYC	AF	NYC	AA	(10)
	Paris	NYC	AF	Paris	None	(11)
	Paris	NYC	AF	Lille	AF	(12)

Figure 20 – The product table with labeled rows.

join queries in an interactive scenario, where the user may label a tuple as either positive or negative, we only investigate learnability of join queries from positive and negative examples. We point out, however, that generally our results are easily adapted to learning join queries from positive examples only. Table 4 contains a summary of learnability results.

	Equi-join queries	*Semi-join queries*
Without disjunction	Yes	No
With disjunction	Yes	Yes

Table 4 – Summary of learnability results.

When only conjunctions of equalities are allowed, learning equi-join queries from examples is relatively easy and is based on constructing the minimal equi-join query fitting the positive examples. To construct the minimal equi-join query fitting a set of positive examples it suffices to identify the set of all pairs of attributes for which all tuples have the same value. For instance, the minimal equi-join query fitting the positive examples in the product table in Figure 20 is

$\{(To, City), (Airline, Discount)\}$ i.e., the query Q_2. The construction of the minimal fitting query is sufficient for learning equi-join queries from form positive and negative examples: the negative examples are not necessary for learning and their presence only requires the learning algorithm to verify consistency by making sure that none of the negatively labeled tuples are selected by the minimal equi-join query fitting the positive examples. For instance, the sample in Figure 20 is consistent because the query Q_2 does not select the negative example.

When we move to semi-joins, again using only conjunctions of equality tests, learnability is unlikely to be feasible as the consistency problem is NP-complete. While this results is not necessarily trivial, it is not particularly surprising. Semi-join queries involve projection that essentially obscures the matching record from one table and allows a significant degree of nondeterminism. This nondeterminism can however be handled by allowing disjunctions in the join condition. Indeed, the consistency problem becomes trivial when disjunctions are allowed and learning is achieved by employing generalization operation that basically remove atomic equality tests under the control of negative examples. The same algorithm works for learning equi-join queries with disjunctions.

5.2 Interactive learning

The main motivation for interactive learning is the large size of the product table: because its size is the product of the sizes of the tables of the database, it might easily get very large and it is unrealistic to assume that the user is able to inspect its entire contents. Consequently, we are interested in developing an approach, where the learning algorithm presents to the user carefully selected tuples whose labeling leads quickly to finding the goal query.

To identify the labels that should be presented to the user we investigate quantifying the potential information that labeling a tuple has on the knowledge of the goal query. In particular, we identify tuples that are *uninformative* i.e., labeling them does not improve our knowledge of the goal query. We illustrate this concept on the example of learning of an equi-join query on the product table in Figure 18. Suppose the user chooses the flight from Paris to Lille operated by Air France (AF) and the hotel in Lille, which corresponds to labeling the tuple (3) as positive. We recall the queries

$$Q_1 = \{(To, City)\} \quad \text{and} \quad Q_2 = \{(To, City), (Airline, Discount)\}$$

and observe that both queries are consistent with this labeling because both select the tuple (3). The tuple (4) is uninformative because if the user labels it as positive, the set of consistent queries does not change, while labeling is as negative renders the sample inconsistent. Since we assume that the goal query is captured by the class of equi-join queries with conjunction of equality tests, the user can only label the tuple (4) as negative by mistake. On the other hand, the tuple (8) is not uninformative because labeling it separates the queries Q_1 and Q_2: if the tuple (8) is labeled as positive, Q_2 is removed from consideration because it does not select the tuple whereas labeling the tuple (8) as negative makes Q_1 inconsistent with the sample since Q_1 selects that tuple. Interestingly, finding whether or not a tuple is uninformative can be done with the help of consistency test because a tuple is uninformative if and only if there is only one way to label the tuple without making the sample inconsistent.

Finding the tuples that allow the fastest convergence towards the goal query, without knowing the goal query, can be defined as a two-player game and an optimal behavior can be determined with a min-max tree [RN10]. However, the min-max tree is exponentially large in the size of the database, and while the upper complexity bound for choosing the optimal branch is PSPACE, the lower bound remains an open question but we believe it to be intractable. Consequently, we have proposed a number of efficient heuristics based on exploration of the lattice of predicates

(cf. Figure 19) and an experimental evaluation on synthetic and real-life data proved them to be successful in addressing the task at hand.

5.3 Paradigm for interactive learning on Big Data

The work on interactive learning has led to a novel paradigm for learning queries on Big Data [2], for any type of database and any type of query formalism. It is depicted in Figure 21 and in essence, it consists of iterative selection of an informative fragment of the database and presenting it to the user for labeling while inferring a query at after each iteration.

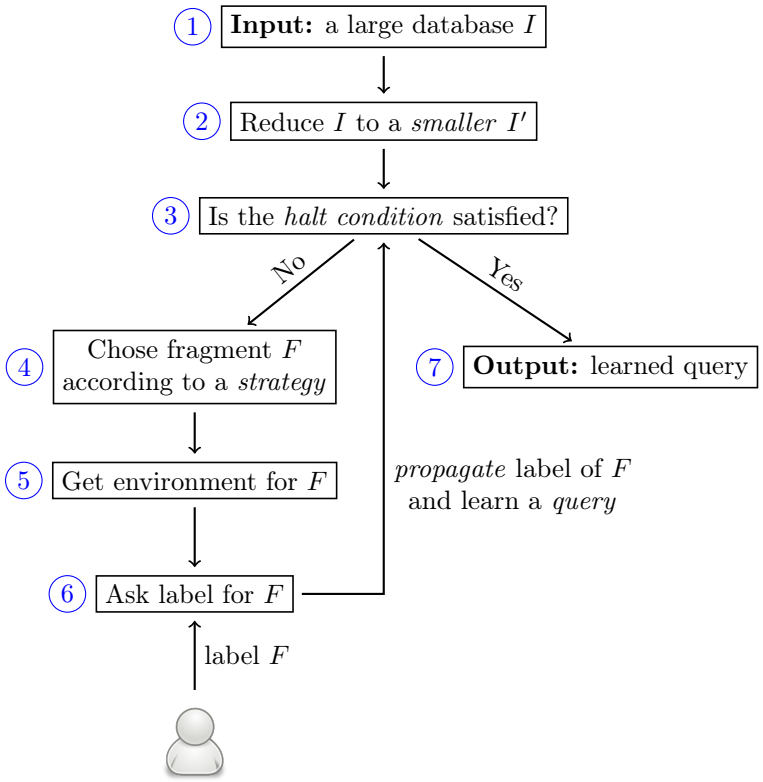

Figure 21 – Workflow of the paradigm.

(1) (2) The paradigm takes as *input* a large database instance I. Because working on the initial instance I might be unfeasible in the first step of preprocessing, an considerably smaller instance I' is constructed that ideally is sufficiently rich to allow identifying the goal query. One way to obtain I' is by removing its redundant fragments. For instance, in the case of join queries, the preprocessing could consist of simply removing the tuples that are equivalent i.e., selected by exactly the same set of queries.

(3) The interactions with the user continue until a *halt condition* is satisfied. A natural halt condition is to stop the interactions when there is exactly one consistent query with the current sample. In practice, we can imagine weaker conditions e.g., the user may stop the process earlier when the current candidate query is deemed satisfactory by the user.

(4) The fragments shown to the user are chosen according to a *strategy* i.e., a function that takes as input an instance I' and a sample S of labeled fragments, and returns a fragment from I'. Since we want to minimize the amount of examples needed to learn the goal query, an intelligent strategy should propose to the user only informative fragments. We point out that while it is possible to design an optimal strategy (i.e., that is guaranteed to propose to the user a minimal number of fragments), such a strategy is usually based on a minimax algorithm [RN10], thus being exponential and unfortunately unfeasible in practice. More practical strategies can be used such as using *entropy* [3] of a fragment that attempts to measure informativeness of a fragment by estimating the possible numbers of queries eliminated from consideration by labeling the fragment.

(5) A fragment by itself does not always carry enough information to allow the user decide how to label it. Therefore, it may be essential to zoom out on the *environment* of a such fragment. While this step for graph databases is relatively nontrivial [BCL15], in case of learning join queries in relational databases it may be even unnecessary (unless, for instance, we deal with a denormalized tables with large numbers of attributes and then, the goal is to identify relevant attributes).

(6) (7) The user visualizes the environment of a given fragment F and labels F appropriately. Then, we propagate the label given by the user for F to the rest of the instance and prune the fragments that become uninformative. Moreover, we run a *learning algorithm* to propose the *best* query that is consistent with all labels provided until this point. When the halt condition is satisfied, we return the last learned query to the user.

6 Transformations

In this section we present our work on learning XML transformations: we have proposed a novel model of *sequential tree-to-word transducers* (STW) [8, 12] and have studied learnability [9] . This required proposing a normal form and an effective methods for *normalization* and *minimization*. It is a significant result because the existence of normal forms for general tree transducers is a long-standing open question [Eng80].

The main motivation to study STWs is that tree-to-word transformations are better suited to model general XML transformations as opposed to tree-to-tree transducers [EMS09, LMN10, MNG08]. This follows from the observation that general purpose XML transformation languages, like XSLT, allow to define transformations from XML documents to arbitrary, not necessarily structured, formats. Also, STWs capture a large subclass of deterministic nested-word to word transducers, which have been the object of a significant interest [FRR+10, SR08]. Because STWs accept ranked trees on the input and produce words on the output, it is not immediately obvious how can they define XML to XML transformations. We point out that it suffices to use the standard first-child next-sibling encoding [CDG+07] to handle XML documents on the input and STWs are capable of producing well-formed serializations of XML on the output.

6.1 Sequential tree-to-word transducers

A *sequential tree-to-word transducer* (STW) is a generalization of the top-down tree automaton [CDG+07] that outputs words while processing the input ranked tree. The definition of an STW consists an *initial rule* of the form $u_0 \cdot q_0(x_0) \cdot u_1$, and a set of *transition rules* of the form $q(f(x_1, \ldots, x_k)) \to u_0 \cdot q_1(x_1) \cdot \ldots \cdot q_k(x_k) \cdot u_k$, where $q, q_0, q_1, \ldots, q_n$ are states, f is an k-ary symbol, $x_0, \ldots, x_k$ are variables, and $u_0, \ldots, u_k$ are words. The transducer is sequential, which means that the order of variables the transition rules is exactly the same on the right-hand side as it is on the left-hand side.

We present an STWs M_1 that serves as a running example; it has the initial rule $q_0(x)$ and the following transition rules:

$$q_0(f(x, y)) \to q_1(x) \cdot adc \cdot q_1(y), \qquad q_1(g(x)) \to q_1(x) \cdot abc, \qquad q_1(a) \to \varepsilon.$$

A run of an STW takes a tree and outputs a word and can be viewed as a rewriting process with the transition rules. We illustrate the run of M_0 on the tree $f(g(g(a)), g(a))$ in Figure 22. In

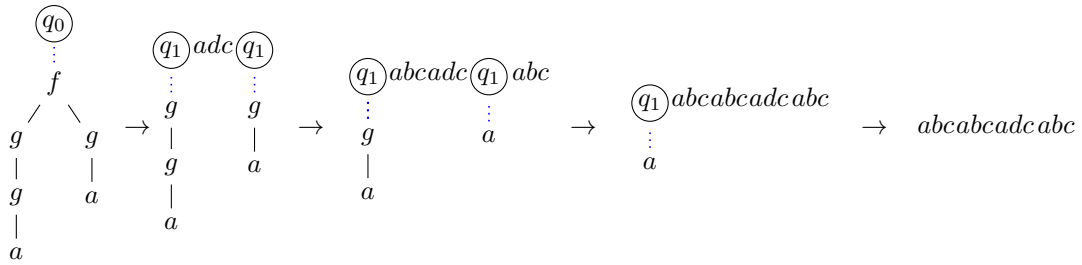

Figure 22 – Example of transformation

general, M_0 defines the following transformation:

$$T_0(f(g^m(a), g^n(a))) = (abc)^m adc (abc)^n.$$

Expressiveness of STWs has two principal limitations: 1) every node is visited exactly once, and 2) the nodes are visited in the fix left-to-right preorder traversal of the input tree. Consequently, STWs cannot express transformations that reorder the nodes of the input tree or make multiple copies of a part of the input tree. STWs remain, however, very powerful and are capable of: concatenation in the output, producing arbitrary context-free languages, deleting inner nodes, and verifying that the input tree belongs to the domain even when deleting parts of it. These features are often missing in tree-to-tree transducers and make STWs incomparable with, for instance, top-down tree-to-tree transducers [EMS09, LMN10].

6.2 Normalization with earliest transducers

In this section we present a normal form for STWs and outline the normalization procedure. The commonly employed method is to require the transducer to produce the output as *early* as possible and its has been successfully applied to wide range of transducers [Cho03, LMN10, FSM10]. However, adapting this method to STWs is a challenging task because the output word is obtained by concatenating the output words in the rules in the standard preorder left-to-right traversal of the tree. As a result it is difficult to define when is the earliest point that a given fragment of the output should be produced. To illustrate this difficulty we present an attempt at defining an earliest transducers that is too eager and results in a normal form that does not cover the whole class of STWs.

Take, for instance, the STW M_{turn} with the initial rule q_{turn} and the following transition rules:

$$\delta(q_{turn}, a) = q_{turn} \cdot a, \qquad \delta(q_{turn}, b) = q_{turn} \cdot b, \qquad \delta(q_{turn}, \bot) = \varepsilon.$$

It defines a transformation T_{turn} that takes a linear tree and output the sequence of its labels in the reverse order e.g., $T_{turn}(a(b(b(\bot)))) = bba$. If we view this transformation as a recursive top-down function that outputs one word upon entering a node and one word upon leaving it, then the earliest moment to produce any output is when the control reaches the leaf $\bot$. At that point the full sequence of labels has been seen and it determines the whole output. This, however, would require storing the sequence of visited nodes in the memory of a transducer, which is finite, and consequently, T_{turn} cannot be expressed with a transducer satisfying this notion of being earliest.

We propose a notion of being *earliest* that is also based on preorder traversal but with the difference that all output words are specified on entering the node but they are not produced on the output until the moment when the control leaves the node. Earliest intuitively means that we wish to *push up* all possible factors in the rules. The STW M_{turn} in the example above satisfies the condition: once the symbol of a node of the input tree is read, the output strings are committed, the control descends, and the strings are produced when the control renters and finally leaves the node. We remark, however, that in some cases the output words in the rule can be placed in several positions, for instance the rule $q_1(g(x)) \to q_1(x) \cdot abc$ in M_0 can be replaced by $q_1(g(x)) \to abc \cdot q_1$ without changing the semantics of M_0. Consequently, we need an additional requirement that resolves this ambiguity: intuitively, we wish to *push left* the words in a rule as much as possible. An STW is *earliest* (eSTW) if $\mathbf{(E_1)}$ for all states no word can be pushed up and $\mathbf{(E_2)}$ if in no word can be pushed left in any of the rules (be it initial or transition).

The transducer M_0 is not earliest because $\mathbf{(E_1)}$ is not satisfied by q_0 and $\mathbf{(E_2)}$ is not satisfied in the rule $q_1(g(x)) \to q_1(x) \cdot abc$. Indeed, every output word produced by the state q_0 begins with a and ends with c, and as we have pointed it out above, abc can be pushed left in the rule $q_1(g(x)) \to q_1(x) \cdot abc$.

The normalization procedure consists of identifying the fragments that can be pushed up and pushed left while appropriately rearranging the output words in the rules. Because different

fragments can be pushed through the same state in a number of different contexts, duplicate copies of the state might need to be created. For M_0, pushing left the abc fragment in the rule $q_1(g(x)) \to q_1(x) \cdot abc$ is relatively harmless as we can simply replace it with the rule $q_1(g(x)) \to abc \cdot q_1(x)$. The trouble starts when we push up a and c in the rule $q_0(f(x,y)) \to q_1(x) \cdot adc \cdot q_1(y)$. Essentially, we need to split the middle output word adc into a, d, and c, and push a left through the first occurrence of the state q_1 while pushing c right through the second occurrence of q_1. Pushing a left through state q_1 introduces a new state q_1' with the following transition rules:

$$q_1'(g(x)) \to bca \cdot q_1'(x), \qquad q_1'(a) \to \varepsilon.$$

Analogously, pushing c right through q_1 introduces a new state q_1'' with the transition rules:

$$q_1''(g(x)) \to cab \cdot q_1''(x), \qquad q_1''(a) \to \varepsilon.$$

Now the rule $q_0(f(x,y)) \to q_1(x) \cdot adc \cdot q_1(y)$ is replaced by

$$q_0'(f(x,y)) \to q_1'(x) \cdot d \cdot q_1''(y),$$

and the a and c are pushed up to the new initial rule $a \cdot q_0'(x_0) \cdot c$.

We point out that the introduction of new states and rules can increase the size of the transducer. In fact, while the normalization procedure works in time polynomial in the size of the constructed earliest transducer, the normalized transducer can be very large and we have shown an exponential lower bound and only doubly-exponential upper bound. We remark that this result is highly nontrivial, as it requires an analysis of combinatorial properties of context-free languages [Lot97], and significant too: the existence of normal forms for more general classes of tree transducers is a long-standing open question [Eng80].

6.3 Minimization with Myhill-Nerode theorem

In order to construct a canonical representative of a transformation defined with an STW we normalize it to an eSTW, which is then minimized. Because of the very rigorous conditions imposed by the normal form, eSTWs can be minimized with a relatively simple adaptation of the standard minimization procedure for deterministic finite automata [HMU01]. However, the definition of earliest STWs allows us to prove a more interesting and more fundamental result, a Myhill-Nerode theorem for STWs, with tractable minimization being only one of its consequences.

The Myhill-Nerode theorem for STWs is based on the notion of residual, which is defined in a mutual recursion with a notion of decomposition. Essentially, for a transformation T and a labeled tree path p the *residual* $p^{-1}T$ is a transformation that captures the operation of T on the subtree of the input tree at the path p. A *decomposition* of T at a given path p is essentially a rule that defines the operation of T at path p with a concatenation of words and output of other transformations applied at the children of the node at p. The definition of decomposition closely mimics the definition of earliest STW and we illustrate these two notions on the example of the transformation T_0 whose definition we recall below

$$T_0(f(g^m(a), g^n(a))) = (abc)^m adc (abc)^n.$$

We first reduce it by essentially by pushing up a and c, and get the following transformation:

$$T_0'(f(g^m(a), g^n(a))) = (bca)^m d (cab)^n.$$

In fact, T_0' is the residual of T_0 at the path ε i.e., $\varepsilon^{-1}T_0 = T_0'$. Now, the decomposition of T_0' at the input symbol f is a tuple $(\varepsilon, T_1', d, T_2', \varepsilon)$, where $T_1'(g^m(a)) = (bca)^m$ and $T_2'(g^m(a)) = (cab)^m$.

We use the name *decomposition* because $T'_0(f(t_1, t_2)) = \varepsilon \cdot T'_1(t_1) \cdot d \cdot T'_2(t_2) \cdot \varepsilon$. Also, T'_1 is the residual of T_0 at the path $(f, 1)$ i.e., $(f, 1)^{-1}T_0 = T'_1$, and analogously $(f, 2)^{-1}T_0 = T'_2$. We continue this process recursively: the decomposition of T'_1 at the symbol g is the tuple (bca, T'_1, ε) while the decomposition of T'_1 at the symbol a is the singleton tuple (ε). Note that the residual of T at path $(f, 1) \cdot (g, 1)$ is again T'_1.

We identify a class of *sequential top-down* transformations as those that have a well-defined residual at every path of their input domain. We also define the *Myhill-Nerode equivalence* on pairs of paths that holds if and only if the corresponding residuals are identical. For instance, for the transformation T_0 this equivalence holds for the two paths $(f, 1)$ and $(f, 1) \cdot (g, 1)$. In fact, for T_0 this equivalence has *finite index* i.e., has a finite number of equivalence classes. By taking the equivalence classes as states and using decomposition to construct rules we can build the *canonical* STW transducer for any sequential top-down transformation with a finite index.

Theorem (Myhill-Nerode for STW) *For any tree-to-word transformation T the following* 3 *conditions are equivalent:*

1. *T is definable with an* STW*,*
2. *T is top-down sequential and has finite Myhill-Nerode index,*
3. *the canonical* STW *of T is the minimal* eSTW *defining T.*

We finish by adding that while minimization for eSTWs is tractable, we show that for STWs it is not: the problem of deciding whether for a given STW there exists an equivalent STW whose size is bounded by a given natural number is NP-complete.

6.4 Learning sequential tree-to-word transducers

The construction of the canonical transducer is defined for functions that are (possibly) infinite sets of pairs of input and output. The learning algorithm for eSTWs is an adaptation of this construction to the setting where the possibly infinite transformation is represented with a finite (characteristic) sample (a finite set of pairs of input and output). It is inspired by the RPNI algorithm [OG91] for inference of regular languages represented with deterministic finite automata. Essentially, the learning algorithm attempts to identify the states and the rules of the goal transducer by constructing residuals of the input sample.

We outline the learning algorithm on the following sample of the transformation T_0, and for clarity we use white space to segment the output string and use bold font for the middle $\boldsymbol{d}$:

$$\begin{aligned} S_0 = \{&(f(a,a), a\boldsymbol{d}c)), (f(a,g(a)), a\boldsymbol{d}c\,abc), (f(a,g(g(a))), a\boldsymbol{d}c\,abc\,abc)\\ &(f(g(a),a), abc\,a\boldsymbol{d}c), (f(g(a),g(a)), abc\,a\boldsymbol{d}c\,abc), (f(g(a),g(g(a))), abc\,a\boldsymbol{d}c\,abc\,abc),\\ &(f(g(g(a)),a), abc\,abc\,a\boldsymbol{d}c), (f(g(g(a)),g(a)), abc\,abc\,a\boldsymbol{d}c\,abc),\\ &(f(g(g(a)),g(g(a))), abc\,abc\,a\boldsymbol{d}c\,abc\,abc)\}. \end{aligned}$$

First, the algorithm reduces the sample: it identifies the longest common prefix of all the output string, which is a, subtracts it from all the output strings, and then identifies and subtracts the longest common suffix, which is c. We get the following reduced sample

$$\begin{aligned} S'_0 = \{&(f(a,a), \boldsymbol{d})), (f(a,g(a)), \boldsymbol{d}\,cab), (f(a,g(g(a))), \boldsymbol{d}\,cab\,cab)\\ &(f(g(a),a), bca\,\boldsymbol{d}), (f(g(a),g(a)), bca\,\boldsymbol{d}\,cab), (f(g(a),g(g(a))), bca\,\boldsymbol{d}\,cab\,cab),\\ &(f(g(g(a)),a), bca\,bca\,\boldsymbol{d}), (f(g(g(a)),g(a)), bca\,bca\,\boldsymbol{d}\,cab),\\ &(f(g(g(a)),g(g(a))), bca\,bca\,\boldsymbol{d}\,cab\,cab)\}. \end{aligned}$$

and the initial rule $(a, [\varepsilon], c)$, where $[\varepsilon]$ denotes the state corresponding to the empty path. We point out that S'_0 is a sample that characterizes T'_0. Next, the algorithm decomposes the sample for the input symbol f into a tuple $(\varepsilon, S'_1, \boldsymbol{d}, S'_2, \varepsilon)$, where

$$S'_1 = \{(a, \varepsilon), (g(a)), bca), (g(g(a)), bca\, bca)\},$$
$$S'_2 = \{(a, \varepsilon), (g(a)), cab), (g(g(a)), cab\, cab)\}.$$

The decomposition is done by essentially observing and identifying common parts in the output string when one of the children of f is fixed while the other varies. For instance, if we restrict S'_0 to $f(x, g(a))$ with varying x i.e., the set

$$\{(f(a, g(a)), \boldsymbol{d}\, cab), (f(g(a), g(a)), bca\, \boldsymbol{d}\, cab), (f(g(g(a), g(a)), bca\, bca\, \boldsymbol{d}\, cab)\},$$

then the longest common prefix of the output strings is ε and their longest common suffix is $\boldsymbol{d}\, cab$. By subtracting them from the output strings we get the sample S'_1. Analogously, the algorithm constructs S'_2, which allows it to identify the output strings ε, $\boldsymbol{d}$, and ε in the decomposition $(\varepsilon, S'_1, \boldsymbol{d}, S'_2, \varepsilon)$ and subsequently the transition rule

$$[\varepsilon](f(x, y)) \to [(f, 1)](x) \cdot \boldsymbol{d} \cdot [(f, 2)](y).$$

The algorithm continues by constructing the decomposition (bca, S''_1) of S'_1 at symbol g, where

$$S''_1 = \{(a, \varepsilon), (g(a), bca)\}.$$

Because $S''_1 \subseteq S'_1$, the algorithm concludes that the paths $(f, 1)$ and $(f, 1) \cdot (g, 1)$ are equivalent, which means that there is no need for the state $[(f, 1) \cdot (g, 1)]$ but instead the state $[(f, 1)]$ should be used. This leads to the rule

$$[(f, 1)](g(x)) \to bca \cdot [(f, 1)](x).$$

Also, the algorithm does not explore any path that is a continuation of the path $(f, 1) \cdot (g, 1)$.

We point out for the algorithm to work well the input sample must be characteristic, i.e., rich enough to allow the construction of all necessary residual samples and decompositions as well as identify equivalence of states reached with pairs of states. If the sample contains all this information, the learning algorithm returns the goal eSTW. However, if the sample is missing any of the necessary information, the algorithm may fail to construct a transducer consistent with the input sample even if the sample is consistent. However, it turns out that testing consistency for STWs is NP-complete, and therefore, it is unlikely that there is a polynomial algorithm that returns a consistent transducer whenever the input sample is consistent. Consequently, we alter the requirements of the learning framework by allowing the learning algorithm to *abstain* from providing a consistent output unless the output contains a characteristic sample.

7 Conclusions and Perspectives

In this dissertation we have presented an overview of a line of research on learning a number of languages for querying, transforming, and describing the structure of databases. While recent advances suggest that natural language interfaces [ART95, LJ14] are likely to play important role in future information systems, we believe that learning from examples lays a foundation for truly adaptive systems. Such systems shall help the users through interaction to formalize their information need without overbearing them with the details of the inner workings of the system. Indeed, it is has been observed that satisfying an information need is a process requiring *negotiation* [Tay68] both from the side of the inquirer and the information provider with the purpose of not only formalizing the information need but also finding out to what degree the information need can be satisfied, and consequently, finding an acceptable *compromise*.

Our research followed the lines of the influential framework of *grammatical inference* [Gol67, dlH05], which is interested in learning algorithms that are *sound* i.e., able to identify a language consistent with the given examples, and *complete* i.e., allowing every language to be identified with a sufficiently informative examples. Soundness implies checking *consistency* of the given set of examples and to certain degree can be seen as a basic diagnostic tool allowing to identify situations where the information needs are too sophisticated for the information system. It is, however, completeness that in our opinion lays foundations to modeling the process of information negotiation. Completeness requires a rigorous understanding of how the search space of a class of languages is organized and how to explore it in a competent manner that allowing to find the desired solution from the user input.

The type of user input that we considered is the most basic one, positive and negative examples. The positive examples illustrate the goal language and often alone allow to correctly infer it. While negative examples can be seen as a way of providing constructive feedback in the negotiation process, our research puts into question whether they fulfill this function adequately. This might be surprising as we do provide a number of algorithms learning from positive and negative examples, however, many of them do not use the negative examples for learning.

In general, negative examples allow to trim down the search space of consistent languages but because our learning algorithms generally work by iteratively improving an expression, the impact that a negative example can have on the expression may be difficult to identify. In context of learning relatively simple languages, such as equi-join queries, negative examples can be very beneficial for improving the knowledge of the goal query. Furthermore, they are essential for learning join queries with disjunction where they control the generation process. However, in the case of a more complex language, such as schemas for Unordered XML, a negative example provides often too little information to benefit learning: if a schema satisfies a negative example, then typically there is a problem with only one of the expressions used by the schema but the negative example does not allow to identify the precise expression that needs to be altered.

These observations are validated by our ongoing work on practical applications of the learning algorithms. Algorithms learning from positive examples generally perform adequately in practice, although there is a problem with overfitting, which we currently try to address. On the other hand, algorithms learning from positive and negative examples, such as learning unions of twig queries, require a significant and very specific sets of negative examples in order to properly control the generalization process. Our results on characterizability further substantiate our belief: to characterize a twig query we need two positive examples and a large number of negative examples.

In our current work on practical inference of schemas for RDF we explore an alternative way of providing negative feedback to the learning algorithm. When an initial candidate schema is constructed by the algorithm, and the input graph is typed against this schema, the user can

identify for a node the types that are currently satisfied by the node but should not be. We also consider input that allows to identify nodes that represent different type of entities, and therefore, should have a different types. However, we point out that the principal challenge in practical learning algorithms for ShEx is scalability: in practice RDF databases can be very large, and our learning algorithms often require at least quadratic time, which very quickly makes our solutions cost prohibitive.

Information needs can typically be closely captured with a language of queries or transformations, and consequently, investigating learning algorithms is a valid approach to lower the accessibility barrier. However, in the future we would like to explore applying our techniques in the context, where the needs of a user are not necessarily closely aligned to any particular formal language, for instance the needs of artistic expression. Our initial research [6] on modeling the structure of music with a generalization of context-free grammars shows that formal languages can be used to model the output of artistic creativity if a degree of fuzziness is introduced. Learning in this context poses a number of interesting challenges such as the choice of an appropriate model of artistic expression. More importantly, this path of research leads to a number of intriguing questions: Is limiting the expression of an artist to a given class of languages while offering active assistance beneficial or detrimental to the creative process? Can machines understand the meaning of taste? Can machines be creative?

Relevant Publications

[1] I. Boneva, R. Ciucanu, and S. Staworko. Schemas for unordered XML on a DIME. *Theoretical Computer Science (TCS)*, 57(2):337–376, 2015.

[2] A. Bonifati, R. Ciucanu, A. Lemay, and S. Staworko. A paradigm for learning queries on Big Data. In *International Workshop on Bringing the Value of Big Data to Users (Data4U)*, pages 7–12, 2014.

[3] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive inference of join queries. In *International Conference on Extending Database Technology (EDBT)*, pages 451–462, 2014.

[4] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive join query inference with JIM. In *International Conference on Extending Database Technology (EDBT)*, pages 1541–1544, 2014. System Demo.

[5] R. Ciucanu and S. Staworko. Learning schemas for unordered XML. In *International Symposium on Database Programming Languages (DBPL)*, 2013.

[6] M. Giraud and S. Staworko. Modeling musical structure with parametric grammars. In *Mathematics and Computation in Music (MCM)*, pages 88–96, 2015.

[7] J. E. Labra Gayo, E. Prud'hommeaux, I. Boneva, S. Staworko, H. R. Solbrig, and S. Hym. Towards an RDF validation language based on regular expression derivatives. In *EDBT/ICDT Workshops (GraphQ & LWDM)*, pages 197–204, 2015.

[8] G. Laurence, A. Lemay, J. Niehren, S. Staworko, and M. Tommasi. Normalization of sequential top-down tree-to-word transducers. In *Language and Automata Theory and Applications (LATA)*, pages 352–363, 2011.

[9] G. Laurence, A. Lemay, J. Niehren, S. Staworko, and M. Tommasi. Learning sequential tree-to-word transducers. In *Language and Automata Theory and Applications (LATA)*, 2014.

[10] J. Michaliszyn, A. Muscholl, S. Staworko, P. Wieczorek, and Z. Wu. On injective embeddings of tree patterns. *CoRR*, abs/1204.4948, 2012.

[11] S. Staworko, I. Boneva, J. E. Labra Gayo, S. Hym, E. G. Prud'hommeaux, and H. Solbrig. Complexity and expressiveness of ShEx for RDF. In *International Conference on Database Theory (ICDT)*, pages 195–211, 2015.

[12] S. Staworko, G. Laurence, A. Lemay, and J. Niehren. Equivalence of deterministic nested word to word transducers. In *International Symposium on Fundamentals of Computation Theory (FCT)*, pages 310–322. Springer LNCS 5699, 2009.

[13] S. Staworko and P. Wieczorek. Learning twig and path queries. In *International Conference on Database Theory (ICDT)*, pages 140–154, 2012.

[14] S. Staworko and P. Wieczorek. Characterizing XML twig queries with examples. In *International Conference on Database Theory (ICDT)*, pages 144–160, 2015.

Bibliography

[AAP+13] A. Abouzied, D. Angluin, C. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 49–60, 2013.

[ABCST92] M. Anthony, G. Brightwell, D. Cohen, and J. Shawe-Taylor. On exact specification by examples. In *Computational Learning Theory (COLT)*, pages 311–318, 1992.

[ABV12] S. Abiteboul, P. Bourhis, and V. Vianu. Highly expressive query languages for unordered data trees. In *International Conference on Database Theory (ICDT)*, pages 46–60, 2012.

[AGP09] M. Arenas, C. Gutierrez, and J. Pérez. Foundations of RDF databases. In *Reasoning Web*, International Summer School on Semantic Technologies for Information Systems, pages 158–204, 2009. Invited Tutorial.

[AHS12] A. Abouzied, J. M. Hellerstein, and A. Silberschatz. Playful query specification with DataPlay. *Proceedings of the VLDB Endowment (PVLDB)*, 5(12):1938–1941, 2012.

[AL05] M. Arenas and L. Libkin. XML data exchange: Consistency and query answering. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 13–24, 2005.

[Ang79] D. Angluin. Finding patterns common to a set of strings. In *ACM Symposium on Theory of Computing (STOC)*, pages 130–141, 1979.

[Ang80] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[APR+11] M. Arenas, J. Pérez, J. Reutter, C. Riveros, and J. Sequeda. Data exchange in the relational and RDF worlds. International Workshop on Semantic Web Information Management (SWIM), 2011. Invited talk.

[ART95] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases – an introduction. *Natural Language Engineering*, 1(01):29–81, 1995.

[AtCKT11a] B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan. Designing and refining schema mappings via data examples. In *ACM SIGMOD International Conference on Management of Data*, pages 133–144, 2011.

[AtCKT11b] B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan. EIRENE: Interactive design and refinement of schema mappings via data examples. *Proceedings of the VLDB Endowment (PVLDB)*, 4(12):1414–1417, 2011.

[AYCLS02] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *VLDB Journal*, 11(4):315–331, 2002.

[Bab64] C. Babbage. *Passages from the Life of a Philosopher*. Longman, Green, Longman, Roberts, & Green, 1864.

[Ban78] F. Bancilhon. On the completeness of query languages for relational data bases. In *Mathematical Foundations of Computer Science (MFCS)*, pages 112–123, 1978.

[BCL15] A. Bonifati, R. Ciucanu, and A. Lemay. Learning path queries on graph databases. In *International Conference on Extending Database Technology (EDBT)*, pages 109–120, 2015.

[BFG08] M Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *Journal of the ACM*, 55(2), 2008.

[BGNV10] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *ACM Transactions on the Web*, 4(4), 2010.

[BGR12] J. Bolleman, S. Gehant, and N. Redaschi. Catching inconsistencies with the semantic web: A biocuration case study. In *International Workshop on Semantic Web Applications and Tools for Life Sciences (SWAT4LS)*, 2012.

[BKW98] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

[BM99] C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In *International Conference on Database Theory (ICDT)*, pages 296–313, 1999.

[BNST06] G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise DTDs from XML data. In *International Conference on Very Large Data Bases (VLDB)*, pages 115–126, 2006.

[BNSV10] G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. *ACM Transactions on Database Systems (TODS)*, 35(2), 2010.

[BNV07] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML schema definitions from XML data. In *International Conference on Very Large Data Bases (VLDB)*, pages 998–1009, 2007.

[BNVdB04] G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: A practical study. In *WebDB*, pages 79–84, 2004.

[CCG06] J. Carme, M. Ceresna, and M. Goebel. Query-based learning of XPath expressions. In *International Colloquium on Grammatical Inference (ICGI)*, pages 342–343, 2006.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[CGLN07] J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1):33–67, 2007.

[CGM15] S. Cebiric, F. Goasdoué, and I. Manolescu. Query-oriented summarization of RDF graphs. In *British International Conference on Databases (BICOD)*, pages 87–91, 2015.

[CGS09] D. Colazzo, G. Ghelli, and C. Sartiani. Efficient inclusion for a class of XML types with interleaving and counting. *Information Systems*, 34(7):643–656, 2009.

[Cho03] C. Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science (TCS)*, 292(1):131–143, 2003.

[CW13] S. Cohen and Y. Y. Weiss. Certain and possible XPath answers. In *International Conference on Database Theory (ICDT)*, 2013.

[CW15] S. Cohen and Y. Y. Weiss. Learning tree patterns from example graphs. In *International Conference on Database Theory (ICDT)*, pages 127–143, 2015.

[Dal00] V. Dalmau. *Computational Complexity of Problems over Generalized Formulas*. PhD thesis, Universitat Politecnica de Catalunya, 2000.

[dlH97] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.

[dlH05] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38:1332–1348, 2005.

[DP92] R. Dechter and J. Pearl. Structure identification in relational data. *Artificial Intelligence*, 58(1):237–270, 1992.

[Edw01] S. M. Edwards. The technology paradox: Efficiency versus creativity. *Creativity Research Journal*, 13(2):221–228, 2001.

[EMS09] J. Engelfriet, S. Maneth, and H. Seidl. Deciding equivalence of top-down XML transformations in polynomial time. *Journal of Computer and System Science*, 75(5):271–286, 2009.

[Eng80] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R. V. Book, editor, *Formal language theory; perspectives and open problems*. Academic Press, 1980.

[FK13] D. D. Freydenberger and T. Kötzing. Fast learning of restricted regular expressions and DTDs. In *International Conference on Database Theory (ICDT)*, pages 45–56, 2013.

[Flo05] D. Florescu. Managing semi-structured data. *ACM Queue*, 3(8):18–24, 2005.

[FLWW12] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *ACM SIGMOD International Conference on Management of Data*, pages 157–168, 2012.

[FRR+10] E. Filiot, J.-F. Raskin, P.-A. Reynier, F. Servais, and J.-M. Talbot. Properties of visibly pushdown transducers. In *Mathematical Foundations of Computer Science (MFCS)*, pages 355–367, 2010.

[FSM10] S. Friese, H. Seidl, and S. Maneth. Minimization of deterministic bottom-up tree transducers. In *Developments in Language Theory (DLT)*, pages 185–196, 2010.

[Gau85] A. Gaur. *A history of writing*. British Library, 1985.

[GCS12] M. C. Gilly, M. W. Celsi, and H. J. Schau. It don't come easy: Overcoming obstacles to technology use within a resistant consumer group. *The Journal of Consumer Affairs*, pages 62–89, 2012.

[GGR+03] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K Shim. XTRACT: Learning document type descriptors from XML document collections. *Data Mining and Knowledge Discovery*, 7(1):23–56, 2003.

[GH66] S. Ginsburg and Spanier E. H. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

[GK95] S. A. Goldman and M. J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20–31, 1995.

[GM11] S. Grijzenhout and M. Marx. The quality of the XML web. In *International Conference on Information and Knowledge Management (CIKM)*, pages 1719–1724, 2011.

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[Gol78] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[GRS93] S. A. Goldman, R. L. Rivest, and R. E. Schapire. Learning binary relations and total orders. *SIAM Journal on Computing*, 22(5):1006–1034, 1993.

[GS10] G. Gottlob and P. Senellart. Schema mapping discovery from data instances. *Journal of the ACM*, 57(2), 2010.

[GVdB09] J. Gillis and J. Van den Bussche. Induction of relational algebra expressions. In *Inductive Logic Programming (ILP)*, pages 25–33, 2009.

[HMU01] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2nd edition, 2001.

[HNW06] J. Hegewald, F. Naumann, and M. Weis. XStruct: Efficient schema extraction from multiple and large XML documents. In *ICDE Workshops*, pages 81–81, 2006.

[JR93] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

[KC10] S. Khatchadourian and M. Consens. ExpLOD: Summary-based exploration of interlinking and RDF usage in the linked open data cloud. *The Semantic Web: Research and Applications*, pages 272–287, 2010.

[KT10] E. Kopczynski and A. W. To. Parikh images of grammars: Complexity and applications. In *Logic in Computer Science (LICS)*, pages 80–89, 2010.

[Kus97] Eyal Kushilevitz. Communication complexity. *Advances in Computers*, 44:331–360, 1997.

[LHB10] W. Lidwell, K. Holden, and J. Butler. *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions*. Rockport Publishers, 2010.

[Lip79] W. Jr. Lipski. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems (TODS)*, 4(3):262–296, 1979.

[LJ14] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment (PVLDB)*, 8(1):73–84, 2014.

[LMN10] A. Lemay, S. Maneth, and J. Niehren. A learning algorithm for top-down XML transformations. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 285–296, 2010.

[Lot97] M. Lothaire, editor. *Combinatorics on Words*. Cambridge Mathematical Library. Cambridge University Press, 2nd edition, 1997.

[MASS08] A. Maduko, K. Anyanwu, A. P. Sheth, and P. Schliekelman. Graph summaries for subgraph frequency estimation. In *European Semantic Web Conference*, pages 508–523, 2008.

[McK97] C. McKinstry. Minimum intelligent signal test: an objective Turing Test. *Canadian Artificial Intelligence*, pages 17–18, 1997.

[MKM04] A. Morishima, H. Kitagawa, and A. Matsumoto. A machine learning approach to rapid development of XML mapping queries. In *International Conference on Data Engineering (ICDE)*, 2004.

[MNG08] W. Martens, F. Neven, and M. Gyssens. Typechecking top-down XML transformations: Fixed input or output schemas. *Information and Computation*, 206(7):806–827, 2008.

[MNS09] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM Journal on Computing*, 39(4):1486–1530, 2009.

[MS04] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51(1):2–45, 2004.

[NS] F. Neven and T. Schwentick. XML schemas without order. 1999.

[NS06] F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, 2(3), 2006.

[OG91] J. Oncina and P. Gracia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, 1991.

[Ong86] W. J. Ong. *Writing is a Technology that Restructures Thought*, pages 23–50. Wolfson College Oxford: Wolfson College lectures. Clarendon Press, 1986.

[Ong02] W. J. Ong. *Orality and Literacy*. New Accents. Taylor & Francis, 2002.

[Opp78] D. C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of presburger arithmetic. *Journal of Computer and System Sciences*, 16(3):323–332, 1978.

[Par66] R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

[Par78] J. Paredaens. On the expressive power of the relational algebra. *Information Processing Letters*, 7(2):107–111, 1978.

[Pea85] R. D. Pea. Beyond amplification: Using the computer to reorganize mental functioning. *Educational psychologist*, 20(4):167–182, 1985.

[QCJ12] L. Qian, M. J. Cafarella, and H. V. Jagadish. Sample-driven schema mapping. In *ACM SIGMOD International Conference on Management of Data*, pages 73–84, 2012.

[RBVdB08] S. Raeymaekers, M. Bruynooghe, and J. Van den Bussche. Learning (k, l)-contextual tree languages for information extraction from web pages. *Machine Learning*, 71(2–3):155–183, 2008.

[RG00] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. WCB/McGraw-Hill, 2000.

[RLHS13] A. Ryman, A. Le Hors, and S. Speicher. OSLC Resource Shape: A language for defining constraints on linked data. In *WWW Workshop on Linked Data on the Web (LDOW)*, 2013.

[RN10] S. J. Russell and P. Norvig. *Artificial Intelligence – A Modern Approach*. Pearson Education, 2010.

[RS91] C. Reutenauer and M. P. Schützenberger. Minimalization of rational word functions. *SIAM Journal on Computing*, 20:669–685, 1991.

[SA95] T. Shinohara and S. Arikawa. Pattern inference. In *Algorithmic Learning for Knowledge-Based Systems*, pages 259–291. Elsevier, 1995.

[Sch04] T. Schwentick. Trees, automata and XML. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 222–222, 2004.

[SDHK91] S. Salzberg, A. L. Delcher, D. G. Heath, and S. Kasif. Learning with a helpful teacher. In *International Conference on Artificial Intelligence (AAAI)*, pages 705–711, 1991.

[Shi82] T. Shinohara. Polynomial time inference of extended regular pattern languages. In *RIMS Symposium on Software Science and Engineering*, pages 115–127, 1982.

[Sir10] E. Sirin. Data validation with OWL integrity constraints. In *International Conference on Web Reasoning and Rule Systems (RR)*, pages 18–22, 2010.

[SM91] A. Shinohara and S. Miyano. Teachability in computational learning. *New Generation Computing*, 8(4):337–347, 1991.

[SPGMW10] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and J Widom. Synthesizing view definitions from data. In *International Conference on Database Theory (ICDT)*, pages 89–103, 2010.

[SR08] F. Servais and J.-F. Raskin. Visibly pushdown transducers. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2008.

[Tay68] R. S. Taylor. Question-negotiation and information seeking in libraries. *College & Research Libraries*, 29(3):178–194, 1968.

[tCDK13] B. ten Cate, V. Dalmau, and P. G. Kolaitis. Learning schema mappings. *ACM Transactions on Database Systems (TODS)*, 38(4), 2013.

[tCKT10] B. ten Cate, P. G. Kolaitis, and W. C. Tan. Database constraints and homomorphism dualities. In *Principles and Practice of Constraint Programming (CP)*, pages 475–490, 2010.

[TCP09] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query by output. In *ACM SIGMOD International Conference on Management of Data*, pages 535–548, 2009.

[TLR13] T. Tran, G. Ladwig, and S. Rudolph. Managing structured and semistructured RDF data using structure indexes. *IEEE Transactions on Knowledge and Data Engineering*, 25(9):2076–2089, 2013.

[TSBM10] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity constraints in OWL. In *International Conference on Artificial Intelligence (AAAI)*, 2010.

[Tur50] A. M. Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[W3C99] W3C. XML path language (XPath) 1.0, 1999. `http://www.w3.org/TR/xpath`.

[W3C04] W3C. OWL web ontology language reference. `http://www.w3.org/TR/owl-ref`, 2004.

[W3C07a] W3C. XML path language (XPath) 2.0, 2007. `http://www.w3.org/TR/xpath20`.

[W3C07b] W3C. XML query (XQuery), 2007. `http://www.w3.org/XML/Query`.

[W3C12] W3C. SPARQL 1.1 entailment regimes. `http://www.w3.org/TR/sparql11-entailment/`, 2012.

[W3C13a] W3C. RDF validation workshop report: Practical assurances for quality RDF data. `http://www.w3.org/2012/12/rdf-val/report`, 2013.

[W3C13b] W3C. Shape expressions schemas, 2013. `http://www.w3.org/2013/ShEx/Primer`.

[WR09] B. Whitworth and H. Ryu. A comparison of human and computer information processing. In M. Pagani, editor, *Encyclopaedia of Multimedia technology and Networking*, pages 230–239. IGI Global, 2nd edition, 2009.

[Wri89] K. Wright. Identification of unions of languages drawn from an identifiable class. In *Workshop on Computational Learning Theory (COLT)*, pages 328–333, 1989.

[WS15] K. Warwick and H. Shah. Can machines think? a report on Turing test experiments at the Royal Society. *Journal of Experimental & Theoretical Artificial Intelligence*, pages 1–19, 2015.

[ZDYZ14] H. Zhang, Y. Duan, X. Yuan, and Y. Zhang. ASSG: adaptive structural summary for RDF graph data. In *International Semantic Web Conference (ISWC)*, pages 233–236, 2014.

[ZEPS13] M. Zhang, H. Elmeleegy, C. M. Procopiuc, and D. Srivastava. Reverse engineering complex join queries. In *ACM SIGMOD International Conference on Management of Data*, pages 809–820, 2013.